NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

OPERATING STANDBY REDUNDANT CONTROLLER TO IMPROVE VOLTAGE SOURCE INVERTER RELIABILITY

by

Stephen T. Blevins

December 2007

Thesis Advisor:	Alexander L. Julian
Second Reader:	Roberto Cristi

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.		
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE Operating Standby Redundant Controller to Improve Voltage Source Inverter Reliability		**5. FUNDING NUMBERS**
6. AUTHOR(S) Stephen T. Blevins		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis presents a control architecture that achieves operating standby redundancy for a voltage source inverter controller. The system was designed to increase reliability by switching from the primary to the secondary controller when a fault to the primary controller occurs. The behavior of the system was predicted using a computer model representing the redundant controller architecture. The simulated results were then verified in lab hardware comprising two FPGAs, a three phase rectifier, an LC filter, and a resistive load. Both simulated and experimental results validate that the final redundant controller design switches between redundant controllers with a negligible disturbance.

14. SUBJECT TERMS Reliability, Voltage Source Inverter, Redundancy, Controller			**15. NUMBER OF PAGES** 147
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

OPERATING STANDBY REDUNDANT CONTROLLER TO IMPROVE VOLTAGE SOURCE INVERTER RELIABILITY

Stephen T. Blevins
Lieutenant, United States Navy
B.S., Clemson University, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2007

Author:	Stephen T. Blevins
Approved by:	Alexander L. Julian Thesis Advisor
	Roberto Cristi Second Reader
	Jeffrey B. Knorr Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis presents a control architecture that achieves operating standby redundancy for a voltage source inverter controller. The system was designed to increase reliability by switching from the primary to the secondary controller when a fault to the primary controller occurs. The behavior of the system was predicted using a computer model representing the redundant controller architecture. The simulated results were then verified in lab hardware comprising two FPGAs, a three phase rectifier, an LC filter, and a resistive load. Both simulated and experimental results validate that the final redundant controller design switches between redundant controllers with a negligible disturbance.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

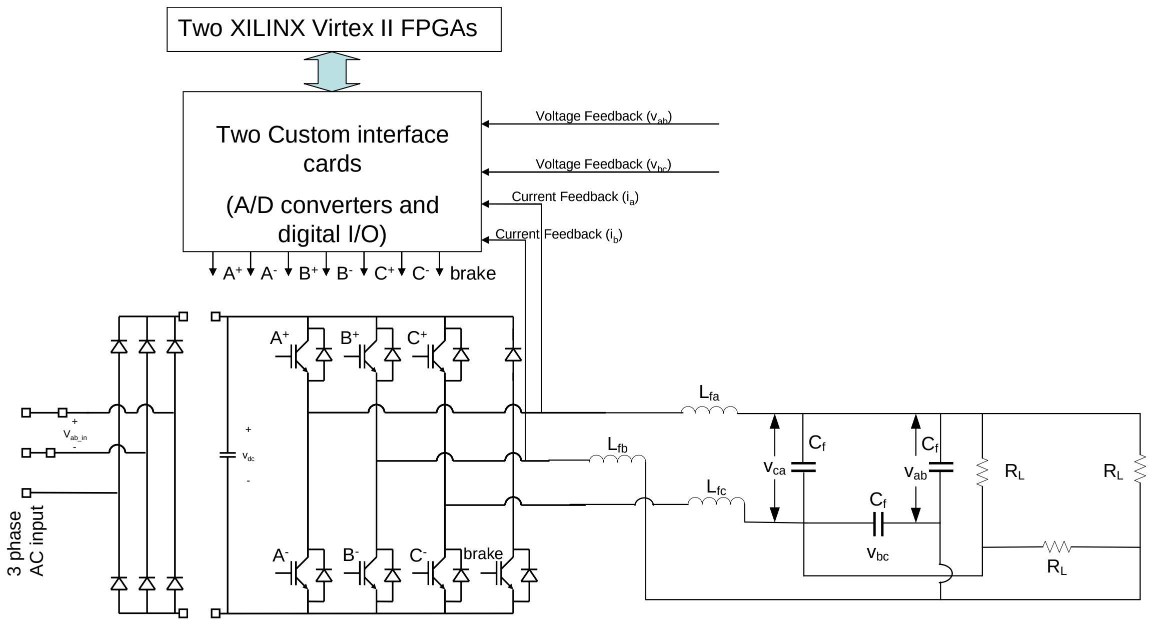
EXECUTIVE SUMMARY

This thesis was conducted to ascertain the feasibility of a strategy to develop a more reliable voltage source inverter (VSI) through operating standby redundant controller architecture. The goal of this research was to determining the level of disturbance in a VSI during a switching event between the primary and redundant controller. Both of the controllers in the VSI for this research were designed based on the closed loop space vector modulation controller developed in the NPS power lab that contained an outer voltage Proportional-Integral (PI) control loop and an inner current PI control loop.

The first objective of this thesis was to design a voltage source inverter with a primary and secondary controller that had a fault signal to act as a failure in the primary controller. When the fault was detected the system would automatically switch from the primary to the secondary controller. The secondary controller would operate physically independent of the primary controller to reduce the risk of damage when a fault occurred, thereby providing true redundancy. Once the ability to switch between controllers was established the second objective was to synchronize both controllers to prevent a random phase shift when a fault occurred. The third and final objective was to enable the secondary controller to begin running with the same internal values as the primary controller had when the fault was detected in order to minimize any disturbance to the VSI output.

The hardware in this project consisted of two Virtex II development kit FPGAs connected to customized interface cards, a three phase rectifier, an LC filter, and a resistive load. The Virtex II FPGA contained the design software that produced the six modulated output signals that went into the six step three phase rectifier. The interface card allowed the two FPGAs to pass information to each other as well as connect with the other hardware components. The three phase rectifier connected to an LC filter with the capacitors in a Delta configuration in order to run a load of three resistors in a Delta configuration. Measurements of v_{ab}, v_{bc}, i_a, and i_b were then fed back into the FPGA via

the interface card in order to produce the hardware configuration of the voltage source inverter with a closed loop control system shown in Figure 1.



Two XILINX Virtex II FPGAs
Two Custom interface cards
(A/D converters and digital I/O)
Voltage Feedback (v_{ab})
Voltage Feedback (v_{bc})
Current Feedback (i_a)
Current Feedback (i_b)
A^+ A^- B^+ B^- C^+ C^- brake
3 phase AC input
v_{ab_in}
v_{dc}
L_{fa}
L_{fb}
L_{fc}
C_f
v_{ca}
v_{ab}
v_{bc}
R_L

Figure 1. Diagram of Hardware Configuration.

The controller software had been previously designed in the NPS power lab using XILINX blocks to create the VHDL code for the FPGA and SIMULINK blocks to model the behavior of the hardware components external to the FPGA and interface boards in order to produce the computer simulations. There were several software additions and modifications that had to be made to the controller design in order to achieve the research objectives. The software used to create the redundant controller architecture consisted of a primary and a secondary controller block, an A to D converter block that read the feedback from the system, and a switching unit subsystem that switched between controllers when a fault was sensed. SIMULINK blocks were used to simulate the behavior of the three phase rectifier, LC filter, and resistive load in order to produce predictive simulation results prior to loading the software on the FPGAs. The software picture of the final design is shown in Figure 2.

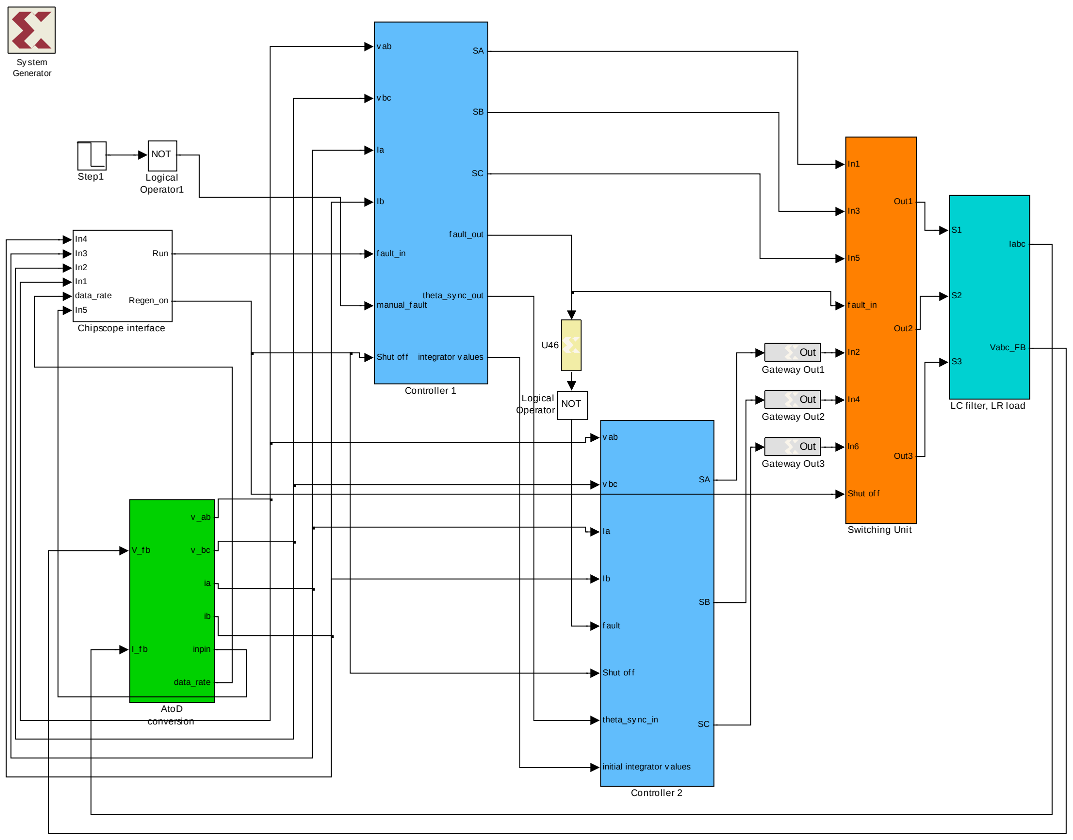


Figure 2. XILINX Model of the Redundant Controller Design.

The first configuration of the redundant controller design only passed the fault signal from the primary to the secondary controller, which produced considerable disturbance in both the phase and amplitude of the VSI output during the switching event. Once the ability to switch between the two controllers had been achieved the next step was to try and eliminate the disturbance in the output due to the switching. The solution to eliminate the random phase shift observed in the output was to send a synchronization signal from the primary controller to the secondary controller. The synchronization signal stopped the internal values of the two controllers from drifting apart over time due to the independent clocks on both boards. This additional communication between the two controllers produced an output that maintained its phase during the switching event but still had amplitude disturbance.

The solution for minimizing the amplitude disturbance was to pass the integrator values in the PI control loops of the primary controller to the secondary controller so they could be used as initial conditions for its integrator values when a fault occurred. Until this point in the design it was necessary to keep the secondary controller’s integrator values set to zero to prevent it from coming online at some random point and potentially damaging the VSI. However, keeping the integrator values at zero prior to sensing a fault in the system meant that the VSI output would have to start at zero and transition to steady state, which caused the disturbance in the amplitude of the output.

The process of sending data between two physically independent FPGAs added an extra degree of difficulty to the solution. In order to send a binary value out of the FPGA there had to be one FPGA pin assigned for each bit of the value. Therefore, a solution to serialize and concatenate the bits of the four integrator values of the primary controller was developed. This allowed the four values to be sent from the primary controller across a single bit output and received by the secondary controller by a single bit input. Once the serialized data was in the secondary board it was deserialized into the four separate values through the deserializtion portion of the software and sent to the appropriate PI control blocks. Both the simulated and experimental results of this design showed virtually no disturbance in the VSI output during the switching event when the synchronization signal and the serialized initial conditions were passed from the primary to the secondary controller.

The VSI used in this thesis was designed using computer simulations that were then confirmed through experimental results at each stage of the research. This thesis successfully showed the ability of a VSI to sense the failure of the primary controller and switch to the secondary controller without any disturbance in the voltage output. Achieving these objectives demonstrated the potential for the reliability of a VSI to be significantly improved through the implementation of operating standby redundant controller architecture. Confirming the ability to seamlessly switch from one controller to another while the system was in operation provides a basis for further development of a robust redundant architecture for a VSI. Some follow on research that would be required includes:

- Reducing the distortion in the VSI output due the gain values and noise in the hardware design.
- Implementation of additional redundant components such as a four-switch pole inverter topology.

As our world becomes increasingly dependent on technology, the need to power that technology with fewer interruptions is also increasing. The redundancy design presented in this thesis was shown to be an effective approach to increase the power supply reliability in both military and civilian industry.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like thank Professor Julian for his guidance and instruction during my thesis research and throughout my time here at NPS. His exceptional teaching style has been instrumental in every facet of my graduate education.

I would also like to thank Lieutenant Kenya Williamson and Captain Joe O’Conner for always making themselves available to discuss ideas and give their advice.

Above all I would like to thank my wife Lisa for her love and support in everything I do. Without you all of my accomplishments would be meaningless.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

This thesis was conducted to ascertain the feasibility of a strategy to develop a more reliable voltage source inverter (VSI) through operating standby redundant controller architecture. The goal of this research was to determine and minimize the level of disturbance in the output of a VSI during a switching event from the primary to the redundant controller. Both of the controllers in the VSI for this research were designed as closed loop space vector modulation controllers that contained an outer voltage Proportional-Integral (PI) control loop and an inner current PI control loop. The research conducted on this VSI configuration used computer simulations and experimental measurements to demonstrate the ability of the system to switch from a primary to a secondary controller upon sensing a fault single. The results of these experiments also demonstrated the amount of disturbance to the system output during the switching event showing its potential for use in mission critical systems for both the military and civilian industry.

The reduction in the output disturbance was a critical component of this research. If the output disturbance during the switching event from the primary to the secondary controller could not be reduced to an acceptable level, the redundant controller architecture would not be a viable way to increase reliability. According to MIL-STD-1399 section 300A, the maximum departure voltage ranges from plus or minus 6 to plus or minus 2.5 percent, and the worst case voltage excursion from nominal user voltage ranges from plus or minus 20 to plus or minus 5.5 percent depending on the type of equipment being operated [1].

Increasing the reliability in electronic power supplies through redundant architectures has obvious benefits for combat or shipboard systems that need to stay online during critical operations. Any vital electronic system used for either military or civilian applications would benefit from a power supply with increased reliability. Although this research only dealt with one redundant controller in order to demonstrate the effects of the switching on the VSI output, the techniques in this research could also be applied to designs with additional redundant components. All of the solutions in this thesis were first implemented and evaluated using computer simulations, and then physical models were built in order to produce real world experimental data.

B. OBJECTIVES AND APPROACH

The first objective of this thesis was to design a voltage source inverter with a primary and secondary controller that had a fault signal to simulate a failure in the primary controller. When the fault was detected the system would automatically switch from the primary to the secondary controller. The secondary controller would operate physically independent of the primary controller to reduce the risk of damage when a fault occurred, thereby providing true redundancy. Once the ability to switch between controllers was established the second objective was to synchronize both controllers to prevent a random phase shift when a fault occurred. The third and final objective was to enable the secondary controller to begin running with the same internal values as the primary controller had when the fault was detected in order to minimize any disturbance to the VSI output.

The basic design layout for this thesis to achieve all of the objectives is shown in Figure 3. The controllers were loaded on two separate boards with three physical connections going from the primary to the secondary controller. The first connection was to pass the fault signal, which would initialize the integrators of the secondary controller when the primary controller failed. The fault signal was also sent from the primary controller to the switching unit in order to switch the output gate signals from the primary to the secondary controller. Placing the controllers on separate boards with a fault signal connecting the primary controller, secondary controller, and switching unit achieved the first objective of the research. The second connection sent a pulse signal to keep the internal values of the two controllers synchronized. This connection from the primary to the secondary controller achieved the second objective. The third connection was to pass the values of the primary controller’s integrators in order to provide a starting value for the secondary controller’s integrators. This third connection from the primary to the secondary controller achieved the third objective of the research by enabling the secondary controller to start with the same internal values that the primary controller had when the fault was detected.

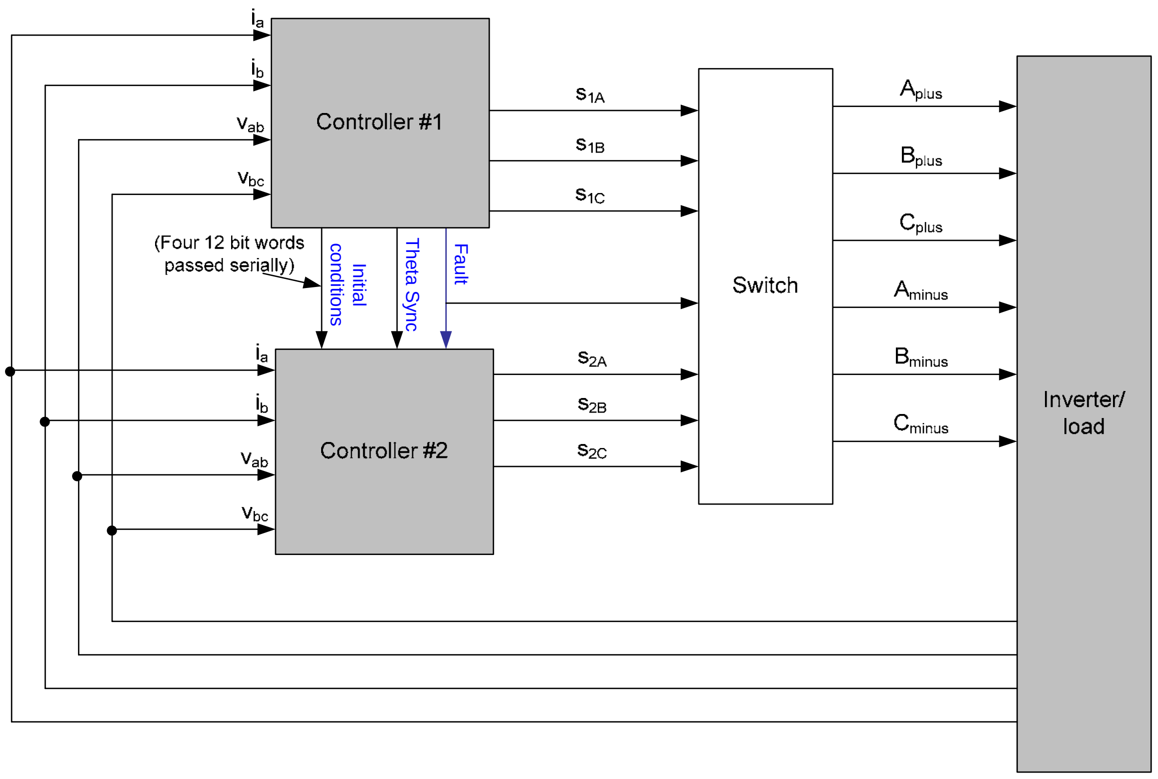


Figure 3. Block Diagram of Redundant Controller VSI.

The redundant controller design was built with two FPGAs programmed using XILINX System Generator and ISE foundation with a discrete algorithm representing the controller architecture shown in Figure 3. XILINX blocks in SIMULINK were used to design the software that was loaded on the two FPGA boards which enabled the simulated behavior of the system to be observed prior to loading the software on the boards. At each stage of the research the software simulation was run in SIMULINK, and the output voltage was observed. Based on the simulated observation, the software was adjusted accordingly until it produced the desired behavior. Once the simulation results were acceptable the software was loaded in the FPGAs and experimental measurements were taken using oscilloscopes and Chipscope software from XILINX. The experimental results were then compared with the software simulations to evaluate any differences.

C. RELATED WORK

The issue of power source reliability has always been a subject of concern for power engineers, and there has been a great deal of research done on different methods of fault detection and redundancy to help increase reliability. A paper presented in 1998 at the International Telecommunications Energy Conference showed evidence which demonstrated the advantages of using modules operating in parallel to achieve increased reliability in dc-ac inverter systems used in uninterrupted power supply systems [2]. Two different redundant configurations, a master-slave and a multi-master, were evaluated to determine the most reliable. The master-slave configuration consisted of one central intelligence module (master) and several local intelligence modules (slaves) that could partly take control of a master failure. The multi-master configuration was designed with all of the modules as independent and equal with full digital control. The multi-master configuration was mathematically shown to be more reliable than the master-slave due to the dependence of all the local intelligence modules on the one centralized master. Therefore, the greater the level of dependency in a redundant inverter system the less reliability the system will have.

Another important issue related to redundancy is the ability to identify and detect the fault modes of an inverter system in order to properly implement the redundant architecture. One such paper explored various fault modes of a voltage-fed Pulse Width Modulation (PWM) inverter system [3]. This research showed an alternative method for increasing reliability in a system as well as the complexity involved in trying to identify specific faults in a system. While it did not address redundant applications, it did demonstrate how the proper detection of a fault in an inverter could help increase reliability by allowing the system to compensate for the fault and operate safely in a degraded mode. An advantage redundant architecture has over fault compensation is the ability to simply shut down the affected component and bring the redundant component online to avoid running the system in a degraded state.

Achieving increased reliability through redundant architectures is also being actively researched for the civilian process industries. A paper concerning reliability of different megawatt drive concepts also discussed some optional redundant designs for a VSI [4]. The redundant approach for that research was to create multiple redundant cellular structures of the Insulated Gate Bipolar Transistor (IGBT) building blocks of the VSI.

These papers on related reliability topics are just a few examples of how diverse this research is in the field of power electronics. The approach taken in this thesis was based on the design concepts presented in reference [5]. In that paper an in depth analysis of the reliability of two different redundant inverter topologies were compared. The redundant four-switch-pole topology was determined to be more reliable than the alternative of a redundant two-switch pole topology. The concept of placing the four-switch-pole inverter in the operating standby redundant controller design was then presented. The block diagram of overall design is shown in Figure 4.

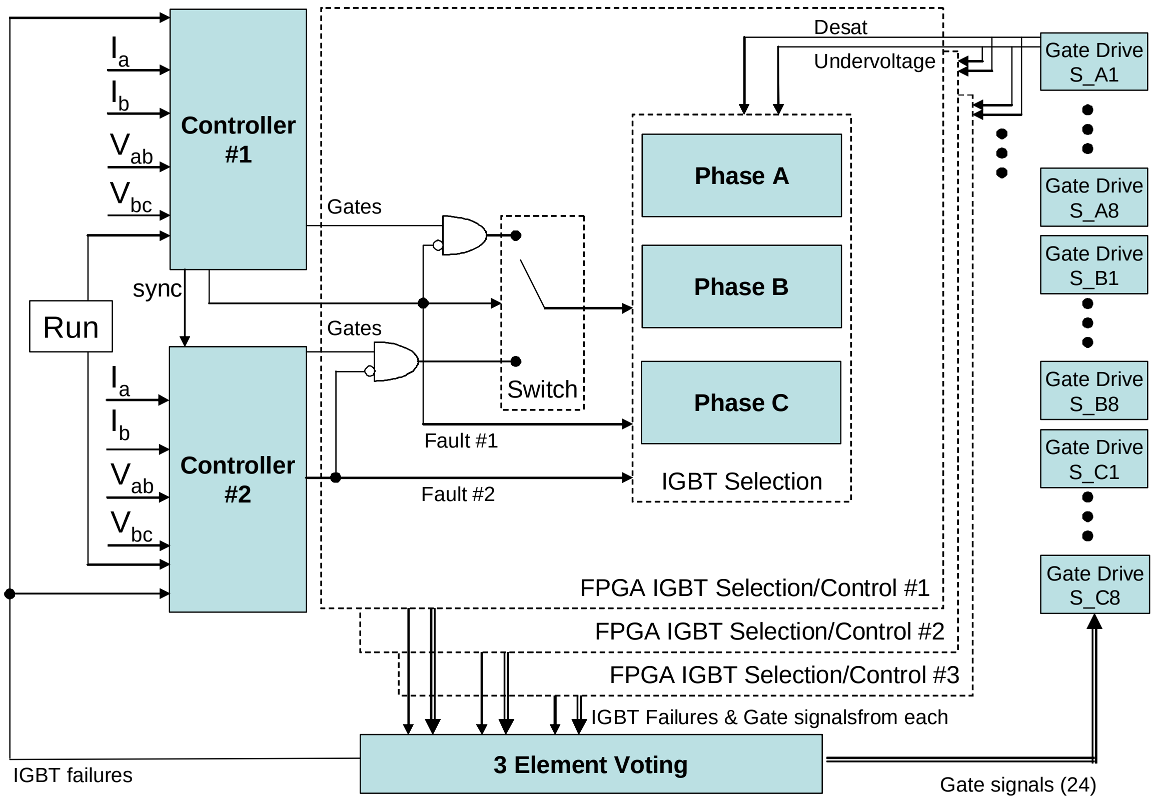


Figure 4. Controller Architecture for Operating Standby Redundancy with Four-Switch Pole Inverter Topology [5].

This thesis took the first steps toward determining the validity of this design by showing the level of disturbance in the VSI output produced from switching to the redundant controller when the primary controller failed. Developing an operating standby redundant controller architecture that would produce little to no disturbance in the output of the VSI was necessary to achieve before this design could be explored further. For this research a regular three phase inverter topology without any switching redundancy was used in order to specifically focus on the controller redundancy.

D. RELIABILITY ANALYSIS

The primary reason for this research was to make a VSI more reliable through the addition of a redundant controller. Therefore, the affects of the additional components on the overall reliability of a VSI should first be quantified. The potential increase in reliability for the VSI can be demonstrated by determining the reliability of each additional component and how the set up of those components affect the overall system using reference [6]. In this thesis the VSI had an additional controller placed in parallel to the original controller, and that parallel configuration was then placed in series with a switching unit. In order to calculate reliability it is first necessary to define the failure rate (λ), which is shown in equation (1.1). Reliability as a function of time is then expressed by equation (1.2), which produces a value greater than 0 and less than 1.

$$\lambda = \frac{\text{Number of failures}}{\text{Total operating hours}} \tag{1.1}$$

$$R(t) = e^{-\lambda t} \tag{1.2}$$

Since the failure of both controllers is required for the system to fail, the primary and secondary controllers are considered to be running in parallel when calculating the reliability of the system. Equation (1.3) shows the calculation of components in parallel with different reliability values, where n represents the number of components in parallel. Equation (1.4) is a modification of equation (1.3) to represent when the reliability of each component is equal. The reliability of components operating in series is shown in equation (1.5), where n represents the number of components in series. Based on these equations the change in reliability from a single controller (R_{contr}) to a redundant configuration with a second controller that has the same reliability and a switching unit in series with a reliability of (R_{su}) can be expressed by equation (1.6).

$$R_{parallel} = 1-(1-R_1)(1-R_2).....(1-R_n) \tag{1.3}$$

$$R_{parallel} = 1-(1-R)^n \tag{1.4}$$

$$R_{series} = (R_1)(R_2).....(R_n) \tag{1.5}$$

$$R_{Total} = R_{su}[1-(1-R_{contr})^2] \tag{1.6}$$

From equation (1.6) it can be shown that the reliability of the VSI has the potential to be significantly increased by placing a redundant controller in parallel with the primary controller. However, the reliability of the system also has the potential to be decreased based on the reliability of the switching unit. These equations demonstrate the trade-offs that must be considered when using redundant components to increase reliability.

E. THESIS ORGANIZATION

The following chapters of this thesis are laid out to provide a clear understanding of what was used in the design process and how the research was conducted. Chapter II gives a detailed description of the software for the closed loop controller used as the foundation for both the primary and secondary controllers in the system. Chapter III describes the hardware components that were chosen for the design and how they interacted. Chapter IV discusses the software design used to produce the basic redundant controller architecture that achieved the first objective of the research. Chapter IV also provides the simulated and experimental results of that design. Chapter V presents the approach used to synchronize the internal values of the two controllers in order to eliminate the phase shift in the VSI output during the switching event which achieved the second objective of the research. The simulated and experimental results for that design are also provided in the chapter. Chapter VI presents the design used to achieve the third and final objective of the research along with the simulated and experimental results that demonstrated the ability of the design to switch with negligible disturbance. Finally, Chapter VII presents the conclusions made based on all the simulated and experimental results of the voltage source inverter designs. This chapter also discusses the potential for follow on work based on this research.

II. CONTROLLER CONFIGURATION

The approach used to achieve the goals for this thesis was based on the controller design chosen to be implemented in the VSI. Both of the controllers in the VSI for this research were based on software that had been previously designed in the NPS power lab to use space vector modulation to modulate the six transistor switches of the three phase rectifier [7]. The closed loop configuration of the controllers contained an outer voltage PI control loop and an inner current PI control loop. The software was designed using the XILINX block set to produce this type of controller and SIMULINK blocks to model the behavior of the hardware components external to the FPGA and interface boards. The XILINX blocks were software additions to SIMULINK that generate the VHDL code required to load the design on the FPGA while the SIMULINK blocks provided a mathematical representation of the hardware in order to produce computer generated simulations of the system. The following sections give a breakdown of the basic controller software design along with the mathematical SIMULINK design that was used to implement the redundant controller architecture simulations for this thesis. For the purposes of this paper, the superscripts e and s represent the synchronous and stationary frames respectively, the subscripts q and d represent the q and d axes, and the subscripts a, b, and c represent the three phases of the voltage and current values.

A. BASIC CONTROLLER DESIGN

The basic design of the controller was the foundation that all of the other software components in this thesis were designed around. This basic configuration was then modified to produce an efficient redundant architecture with a primary controller and one redundant secondary controller. In the design of the closed loop controller the values for the two synchronous frame reference phase voltages (V^e_{qref} and V^e_{dref}) were set to 50 volts and 0 volts respectively by using two constants from the XILINX block library. The controller worked by comparing the reference values to the voltage feedback values in the synchronous frame and sending the results into the first PI controller. The first PI controller then produced reference values for the currents that were compared to the current feedback values in the synchronous frame and sent into the second PI controller. The outputs of the second PI controller were then converted from the synchronous to the stationary frame and sent into the space vector modulation block. The space vector modulation block took the final voltage reference values and translated them into gate signals to be read by the three phase rectifier. Finally the output of the hardware configuration was sent back into the A to D converter to close the loop on the control system. This basic operation for both controllers is laid out in the block diagram in Figure 5.

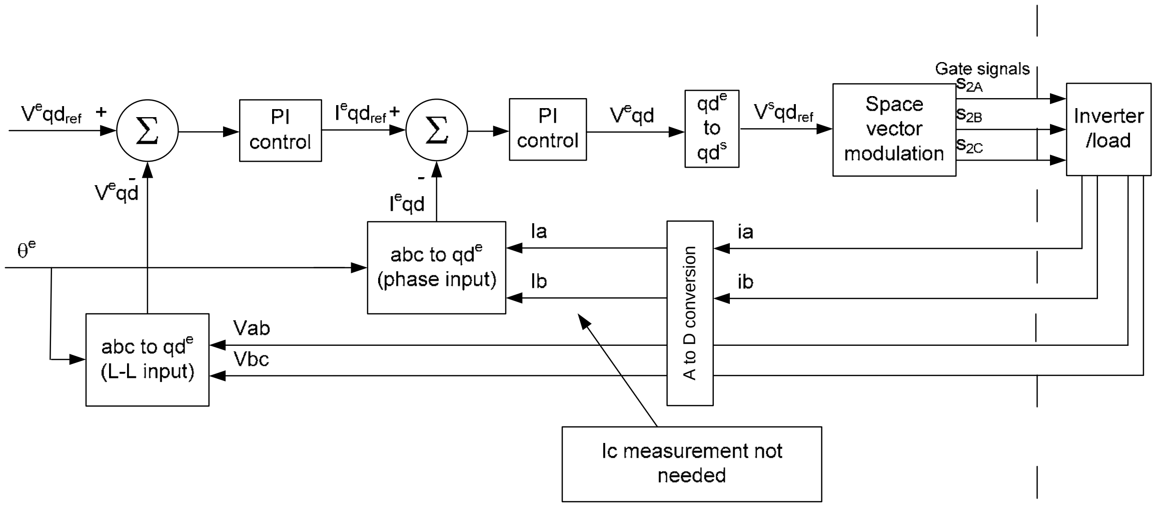


Figure 5. Basic Space Vector Modulation Controller Configuration.

B. THETA DESIGN

The rate of theta was also designed into the software using the XILINX block library. The basic design for the theta value was made up of a constant value of 2π multiplied by the frequency of the system (100 Hz) multiplied by the clock period of the system (40ns). This constant value was sampled every clock period and sent into an accumulator. The accumulator output was then sent back into a rational block that was set to trigger the accumulator to reset when the output value reached 2π. The output of the theta block was then converted from a value of 0 to 2π to a value of 0 to 2^{10} in order to be implemented in the rest of the code. The XILINX configuration used to produce the theta value for the system is shown in Figure 6.

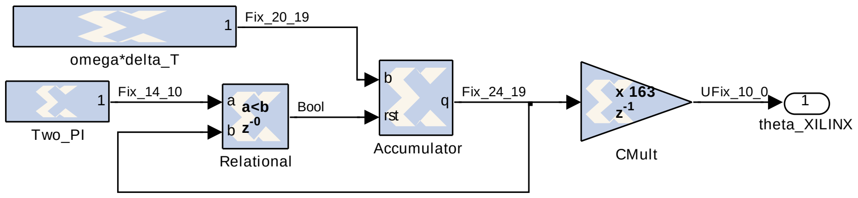


Figure 6. Theta Software Design.

C. FRAME TRANSFORMATIONS

Since the controller used space vector modulation, it was necessary to convert the feedback voltages into the qd frame. The value from the theta design was used in the transformation blocks to convert the V_{ab}, V_{bc}, I_a, and I_b feedback values produced by the A to D converter to the synchronous frame values as shown in Figure 5. The voltage values converted into the synchronous frame (V^e_q and V^e_d) were then subtracted from the set reference values and sent into the voltage PI control block. Similarly the current values converted into the synchronous frame (I^e_q and I^e_d) were subtracted from the reference currents produced by the voltage PI control block and sent to the current PI control block. The final frame conversion block took the new V^e_q and V^e_d values produced by the current PI control block and converted them from the synchronous to the stationary frame.

The equations used to build the abc to qde transformation blocks in the code were derived from equations (2.1) and (2.2), which were taken from reference [8].

$$f_q = \frac{2}{3}\left[f_a \cos(\theta) + f_b \cos\left(\theta - \frac{2\pi}{3}\right) + f_c \cos\left(\theta + \frac{2\pi}{3}\right)\right] \tag{2.1}$$

$$f_d = \frac{2}{3}\left[f_a \sin(\theta) + f_b \sin\left(\theta - \frac{2\pi}{3}\right) + f_c \sin\left(\theta + \frac{2\pi}{3}\right)\right] \tag{2.2}$$

Using the fact that $v_a + v_b + v_c = 0$ and $\text{i}_a + i_b + i_c = 0$ in three phase configurations allowed f_c to be substituted in equations (2.1) and (2.2) yielding the following equations.

$$f_q = \frac{2}{3}\left[f_a \cos(\theta) + f_b \cos\left(\theta - \frac{2\pi}{3}\right) + (-f_a - f_b)\cos\left(\theta + \frac{2\pi}{3}\right)\right] \tag{2.3}$$

$$f_d = \frac{2}{3}\left[f_a \sin(\theta) + f_b \sin\left(\theta - \frac{2\pi}{3}\right) + (-f_a - f_b)\sin\left(\theta + \frac{2\pi}{3}\right)\right] \tag{2.4}$$

These equations can be further simplified to yield

$$f_q = \frac{2}{3}\left[f_a\left(\cos(\theta) - \cos\left(\theta + \frac{2\pi}{3}\right)\right) + f_b\left(\cos\left(\theta - \frac{2\pi}{3}\right) - \cos\left(\theta + \frac{2\pi}{3}\right)\right)\right] \tag{2.5}$$

$$f_d = \frac{2}{3}\left[f_a\left(\sin(\theta) - \sin\left(\theta + \frac{2\pi}{3}\right)\right) + f_b\left(\sin\left(\theta - \frac{2\pi}{3}\right) - \sin\left(\theta + \frac{2\pi}{3}\right)\right)\right] \tag{2.6}$$

A final simplification using trigonometric identities yields the equations used to design the transformation block for the currents in the software.

$$i_q = \frac{2}{3}\left(i_a \sin\left(\theta + \pi/3\right) + i_b \sin(\theta)\right) \tag{2.7}$$

$$i_d = \frac{2}{3}\left(-i_a \cos\left(\theta + \frac{\pi}{3}\right) - i_b \cos(\theta)\right) \tag{2.8}$$

The XILINX block software that corresponds to equations (2.7) and (2.8) is shown in Figure 7.

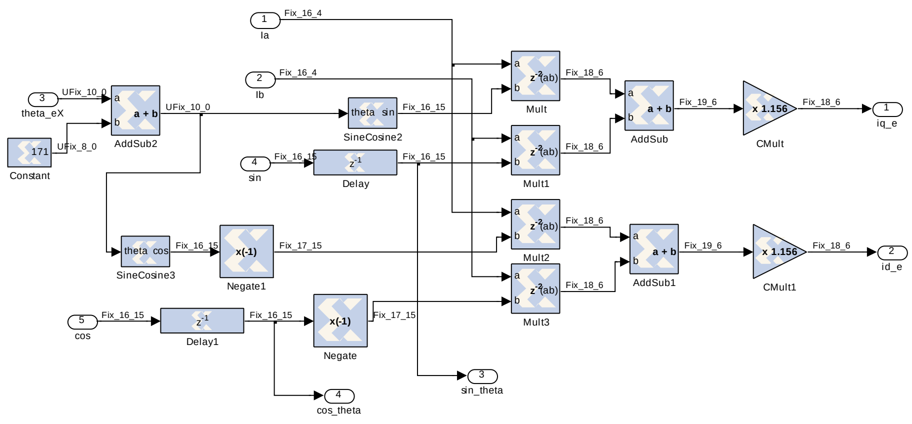


Figure 7. Current Transformation Block.

The line-to-line voltage transformations were derived by using the fact that $v_{ab} = v_a - v_b$ and $v_{bc} = v_b - v_c = v_b - (-v_a - v_b)$ can be manipulated to produce

$v_a = \frac{2}{3}v_{ab} + \frac{1}{3}v_{bc}$ and $v_b = -\frac{1}{3}v_{ab} + \frac{1}{3}v_{bc}$ which, when substituted in equations (2.7) and (2.8) yields

$$f_q = \frac{2}{3\sqrt{3}}\left(2 f_{ab}\sin\left(\theta + \frac{\pi}{3}\right) + f_{bc}\sin\left(\theta + \frac{\pi}{3}\right) - f_{ab}\sin(\theta) + f_{bc}\sin(\theta)\right) \tag{2.9}$$

$$f_q = \frac{2}{3\sqrt{3}}\left(-2 f_{ab}\cos\left(\theta + \frac{\pi}{3}\right) - f_{bc}\cos\left(\theta + \frac{\pi}{3}\right) + f_{ab}\cos(\theta) - f_{bc}\cos(\theta)\right) \tag{2.10}$$

These equations can be simplified further to produce

$$f_q = \frac{2}{3\sqrt{3}}\left(f_{ab}\left(2\sin\left(\theta + \frac{\pi}{3}\right) - \sin(\theta)\right) + f_{bc}\left(\sin\left(\theta + \frac{\pi}{3}\right) + \sin(\theta)\right)\right) \tag{2.11}$$

$$f_q = \frac{2}{3\sqrt{3}}\left(f_{ab}\left(-2\cos\left(\theta + \frac{\pi}{3}\right) + \cos(\theta)\right) - f_{bc}\left(\cos\left(\theta + \frac{\pi}{3}\right) + \cos(\theta)\right)\right) \tag{2.12}$$

A final simplification using trigonometric identities yields the equations used to design the transformation block for the voltages in the software

$$v_q = \frac{2}{3}\left(v_{ab}\cos(\theta) + v_{bc}\sin\left(\theta + \frac{\pi}{6}\right)\right) \tag{2.13}$$

$$v_q = \frac{2}{3}\left(v_{ab}\sin(\theta) + v_{bc}\cos\left(\theta + \frac{\pi}{6}\right)\right) \tag{2.14}$$

The XILINX block software that corresponds to equations (2.13) and (2.14) is shown in Figure 8.

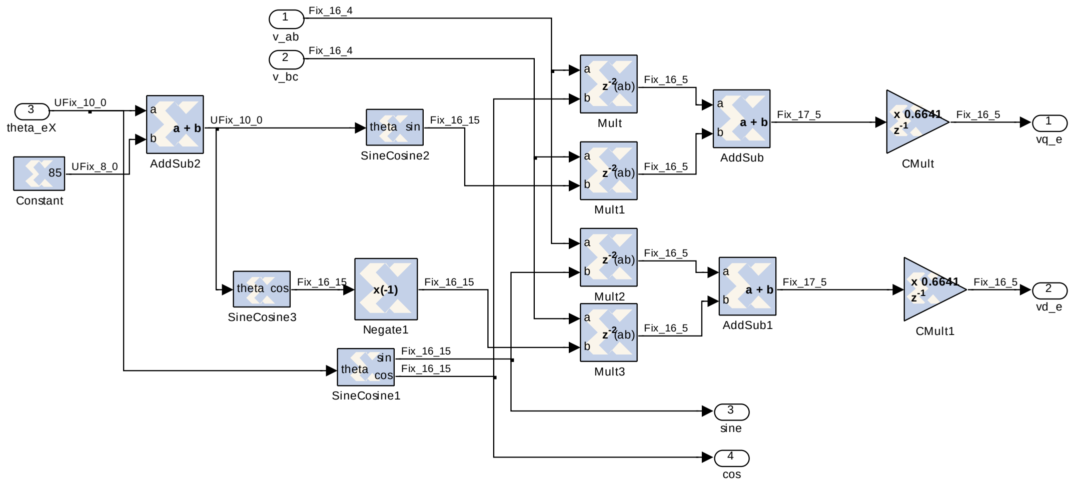


Figure 8. Voltage Transformation Block.

The transformation block that transforms the voltage values from the synchronous to the stationary frame was designed based on the following equation from reference [8].

$$\begin{bmatrix} v_{qs} \\ v_{ds} \end{bmatrix} = \begin{bmatrix} \cos(\theta_e) & \sin(\theta_e) \\ -\sin(\theta_e) & \cos(\theta_e) \end{bmatrix} \begin{bmatrix} v_{qe} \\ v_{de} \end{bmatrix} \tag{2.15}$$

The XILINX representation of equation (2.15) is shown in Figure 9.

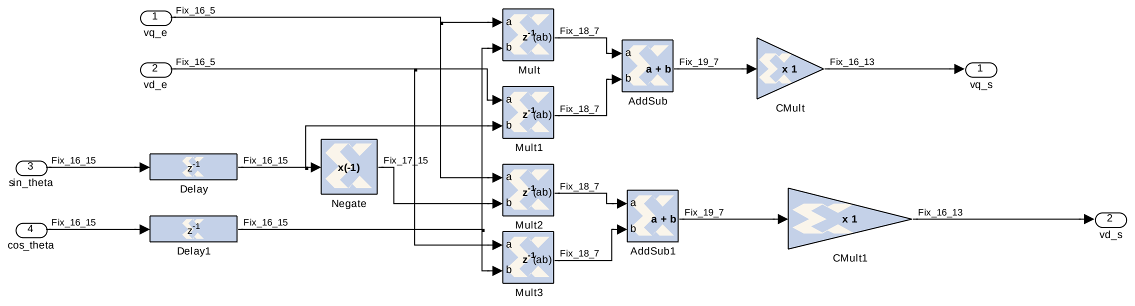


Figure 9. Synchronous to Stationary Frame Transformation.

D. PI CONTROLLERS

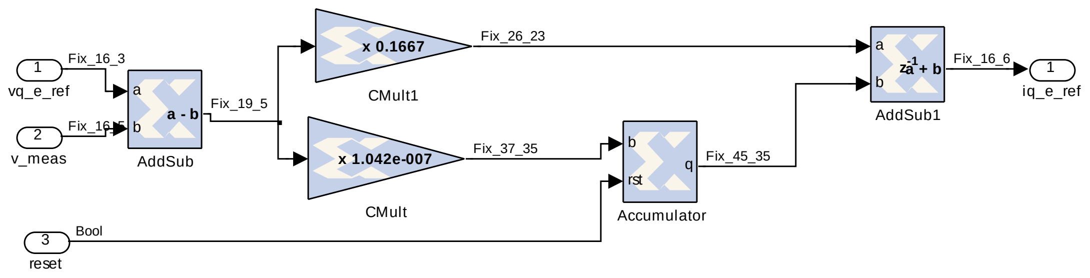


Figure 10. Software for V_q PI Control Block.

All four PI controllers had the same basic design which consisted of the reference value being subtracted by the corresponding measured value. That value was then sent to two separate gain operators to produce a product and an integrator value. The integrator value was sent into an accumulator, and the accumulated value was summed with the product value to produce the appropriate reference output. The controller software had a PI control block with the same design for each of the four variables: V_q, V_d, I_q, I_d. The basic design of all four PI controllers is represented in this section by the V_q PI control block shown in Figure 10.

E. SPACE VECTOR MODULATION

The controller design selected for this thesis was based on space vector modulation to produce the output gate signals for each controller that would modulate the six IGBTs in the three phase rectifier based on the input of the qd voltage values in the stationary qd frame presented in Figure 9. The qd values in the stationary frame were converted to Polar coordinates before being sent into the space vector modulation block. The XILINX block configuration that produced the space vector modulation for this controller is presented in Figure 11.

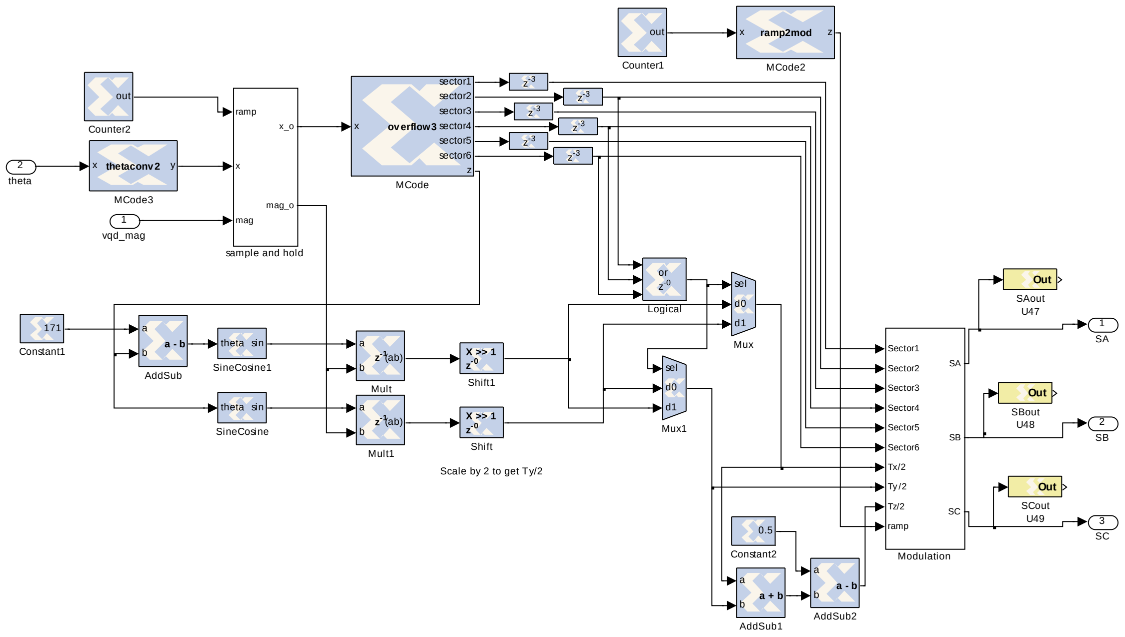


Figure 11. XILINX Block Software for Space Vector Modulation.

The input magnitude and the theta value, that was converted into a binary value, was sent into a sample and hold block that was then sent into an MCode block that selected the appropriate sector of the space vector modulation hexagon. The sector selection outputs were then sent into the modulation block. The additional XILINX blocks in the figure produce the duty cycles that are also sent into the modulation block along with the ramp input. The inputs to the modulation block were sent through numerous arithmetic and logic blocks that produced the three gate signals for the controller [9]. The full set of notes that provide a detailed explanation of the algorithms that describe how the sample and hold block and the modulation block work together with the rest of the design is located in Appendix A.

F. ANALOG TO DIGITAL CONVERTER

The A to D converter for the controller was designed with XILINX blocks to take in the voltage and current feedback signals from the SIMULINK blocks that provided a mathematical representation of real world hardware. While the XILINX blocks were used to create the software code that could be loaded on an FPGA, the SIMULINK blocks were used to represent real world components outside of an FPGA in order to create accurate simulations prior to loading the software on a board. The SIMULINK blocks were also used as pulse generators to simulate the internal clocks on a board. In order for the XILINX blocks to read the SIMULINK blocks in the software the SIMULINK blocks must be sent into one of the yellow input blocks seen in Figure 12. The data into an input block could be a Boolean, signed (2's complement), or unsigned data. However, the input block had to assign one FPGA pin for every data bit. The grey output blocks represent outputs that did not have an FPGA output pin assigned and were used to send information back to the SIMULINK blocks. Once the SIMULINK data was read into the XINLINX blocks of the A to D converter, the timing from the simulated clock values were used to select the two current and two voltage values that were then sent into the controller. The multiplication blocks prior to the output pins were used to offset any scaling of the feedback signals from the hardware when the software was actually loaded on the FPGA.

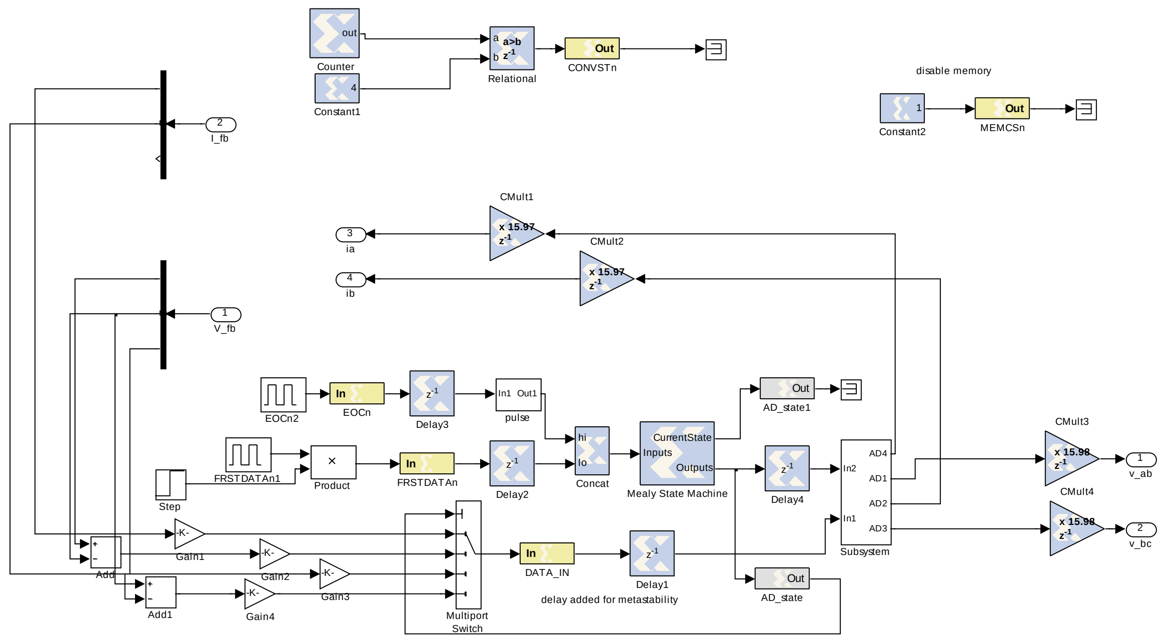


Figure 12. XILINX A to D Converter Design.

G. SIMULATED HARDWARE DESIGN

The software design in SIMULINK that was used to mathematically simulate the external hardware of the VSI in order to create a computer generated output prior to loading the software on the FPGA is shown in Figure 13. The modulated gate signals from the control were sent into this block and mathematically manipulated to produce an output that simulated the gate signals going through a LC filter, with the capacitors in a wye configuration and an LR load in a wye configuration. The values of the simulated filter inductors and capacitors were $350\,\mu\text{H}$ and $60\,\mu\text{F}$ respectively. The values of the simulated load resistors were $20\,\Omega$. Since the hardware design for this research only used a resistive load, the inductor values in the simulated load were set at $100\,\mu\text{H}$ to account for the inductance in the wires of the hardware.

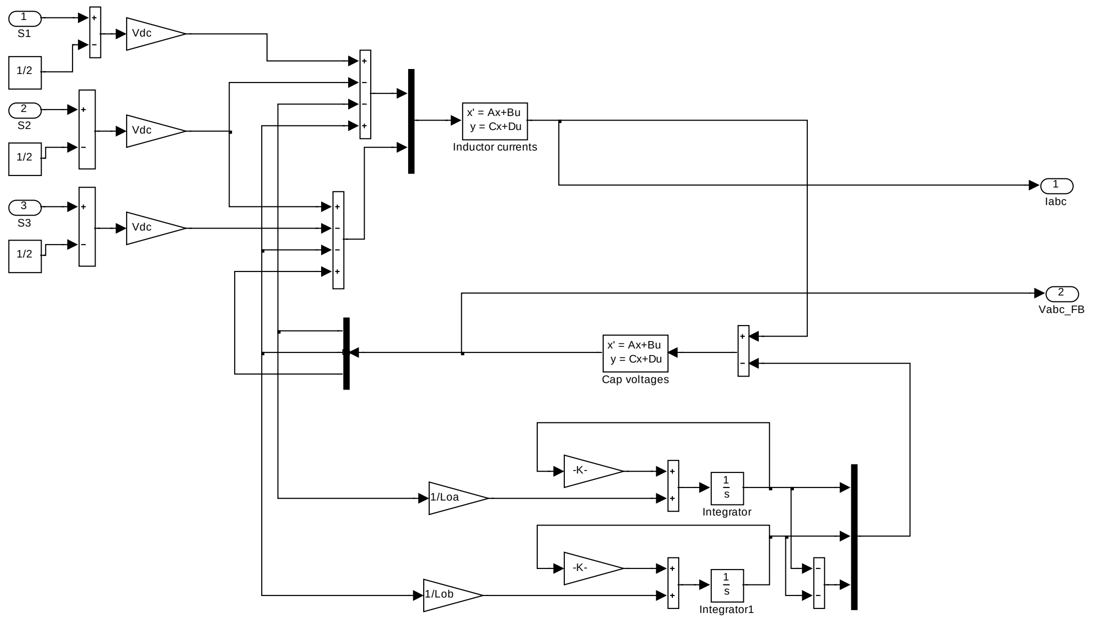


Figure 13. Simulated Hardware Design.

H. CHIPSCOPE

A Chipscope interface block was also a previously designed piece of software that was incorporated into the research design but was not a direct component of the controller. The Chipscope interface block was taken directly from a buck converter lab for EC4150 at NPS [10] and is shown in Figure 14. The software was chosen because it was designed to provide two switches that could be controlled through the Chipscope program from XILINX that was loaded on the computer. The Chipscope interface software was implemented as a remote computer based switch to help prevent unnecessary physical contact with the boards that might have lead to unintentional damage to the hardware. The XILINX simulation multiplexer blocks also allowed the Chipscope interface block to operate the step function that was used for the switching event in the computer simulations as well as the computer switches in the experimental tests. One other feature provided by the software in the Chipscope interface block was the ability to read four internal signals and display them in Chipscope. This feature was also taken from the buck converter lab software to provide the ability to take internal measurements of the software without the need to send them to external pins on the FPGA. The code that was used in the black box block of the software is listed in Appendix F.

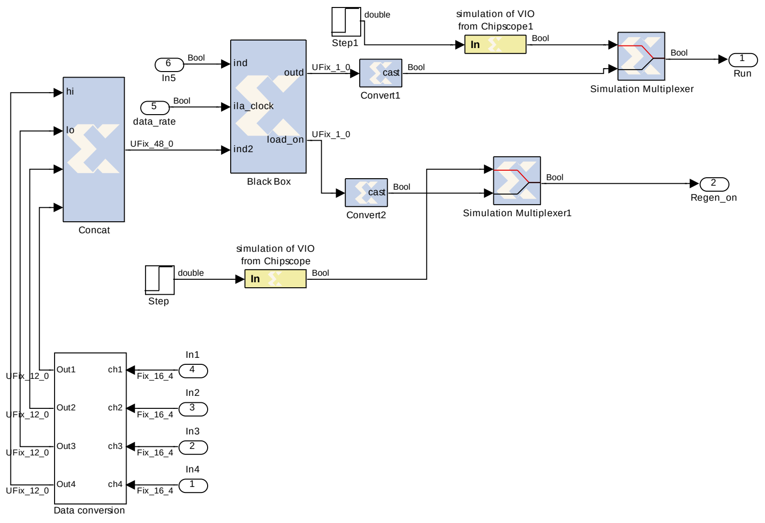


Figure 14. Chipscope Interface Block [10].

I. CHAPTER SUMMARY

This chapter presented the XILINX block components of the basic controller design that were used to implement the redundant controller architecture in this research. The controller was a closed loop designed with an outer voltage PI control loop and an inner current PI control loop. The controller used space vector modulation to modulate its output gate signals. This chapter also showed the way in which SIMULINK and XILINX blocks were used to develop computer simulations that could then be directly transferred to an FPGA for experimental testing. The next chapter presents the hardware configuration that was set up to conduct the experimental testing of the software design.

THIS PAGE INTENTIONALLY LEFT BLANK

III. HARDWARE DESIGN

The next step in the research process was to select hardware components that could be used to conduct real world experiments in order to collect experimental data from the redundant controller design to compare with computer simulated results. The hardware in this project consisted of two Virtex II development kit FPGAs connected to customized interface cards, a three phase rectifier, an LC filter, and a resistive load. The Virtex II FPGA contained the design software that produced the six modulated output signals that went into the six step inverter. The interface card, which included an A/D converter and digital I/O ports, allowed the FPGA to connect with the six step three phase rectifier from SEMIKRON. The rectifier then connected to an LC filter with the capacitors in a Delta configuration in order to run a load of three resistors in a Delta configuration. Measurements of v_{ab}, v_{bc}, i_a, and i_b were then fed back into the FPGA via the interface card in order to produce a voltage source inverter with a closed loop control system as shown in Figure 15.

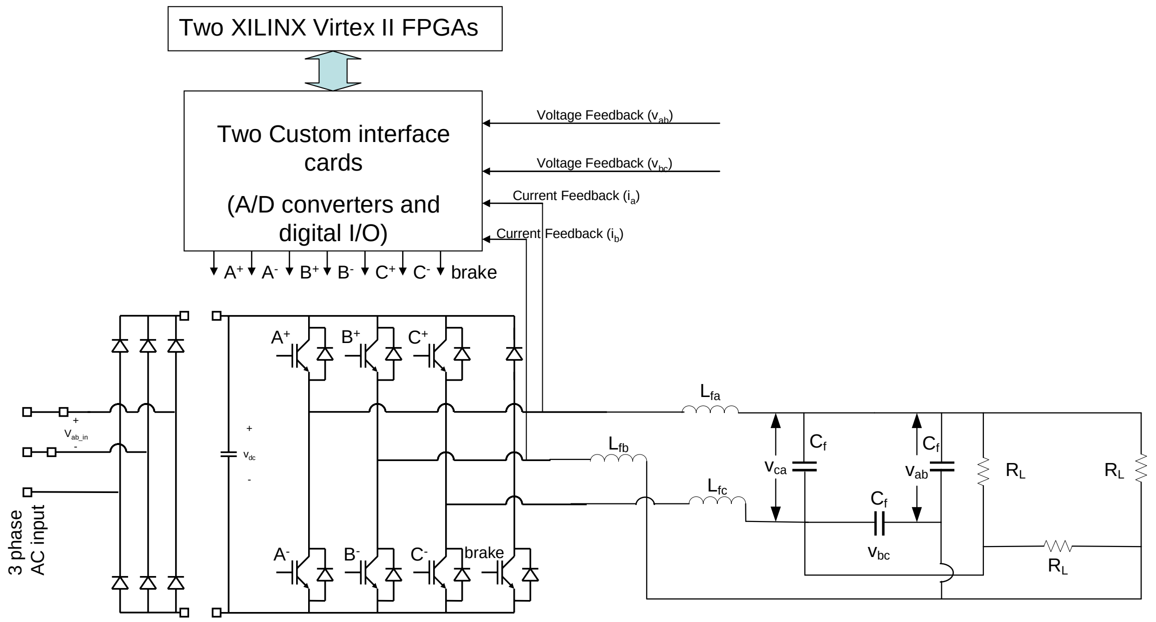


Figure 15. Diagram of the Hardware Configuration.

A. FPGA

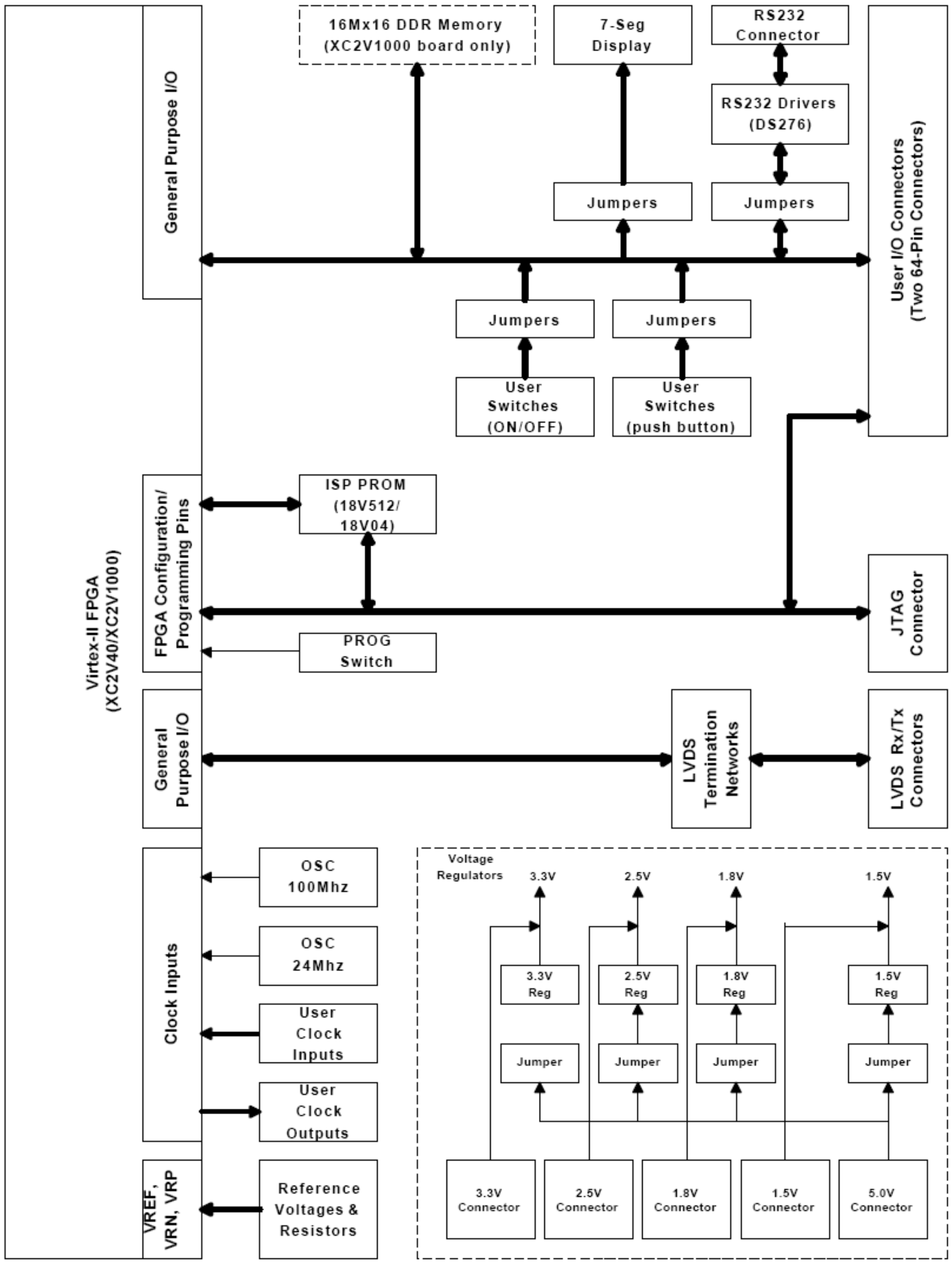


Figure 16. Virtex II High Level Block Diagram [11].

Two Virtex II FPGA boards were used in this research to design the hardware configuration used to collect the experimental data. The boards had been previously purchased and used for other NPS research because of the versatility provided in the development and verification of FPGA designs. The Virtex II provided an Indexed Sequential Processor (ISP) Programmable Read Only Memory (PROM) along with a Joint Test Action Group (JTAG) connector that allowed direct configuration of the FPGA from the computer [11]. A high level block diagram of the FPGA from the Virtex II Reference Board User's Guide is shown in Figure 16.

Using the FPGA to design the controller instead of solid state components provided greater flexibility in making changes to the design along with the ability to load different versions of the design without creating additional boards. The primary board held the software for the primary controller and the switching unit, and the second board held the software for the secondary controller only. The primary FPGA was also used to send the outputs of the selected controller to the three phase rectifier through the switching software located on the primary FPGA. All of the information passed from the primary to the secondary controller had to be sent externally through the interface cards connected to the FPGAs. Likewise the output values of the secondary controller had to be sent back into the primary board externally to the switching unit so it could be passed to the rectifier when a fault occurred. The three modulated outputs from the primary controller and the fault signal from the primary controller were the only information signals passed to the switching unit internally during the experiment. If the design were to be put into practical use, the switching unit should be loaded on its own separate board to avoid a potential failure in the switching unit in the event that the primary controller failure somehow caused damage to the board. However, due to resource limitations only two boards were available, and having the switching unit on the primary board did not have an effect on the measurements of the switching disturbance for the purposes of the experimental research. The product description for the Virtex II board used in the experiments is located in Appendix B. A picture of the Virtex II board used for the experiments is shown in Figure 17.

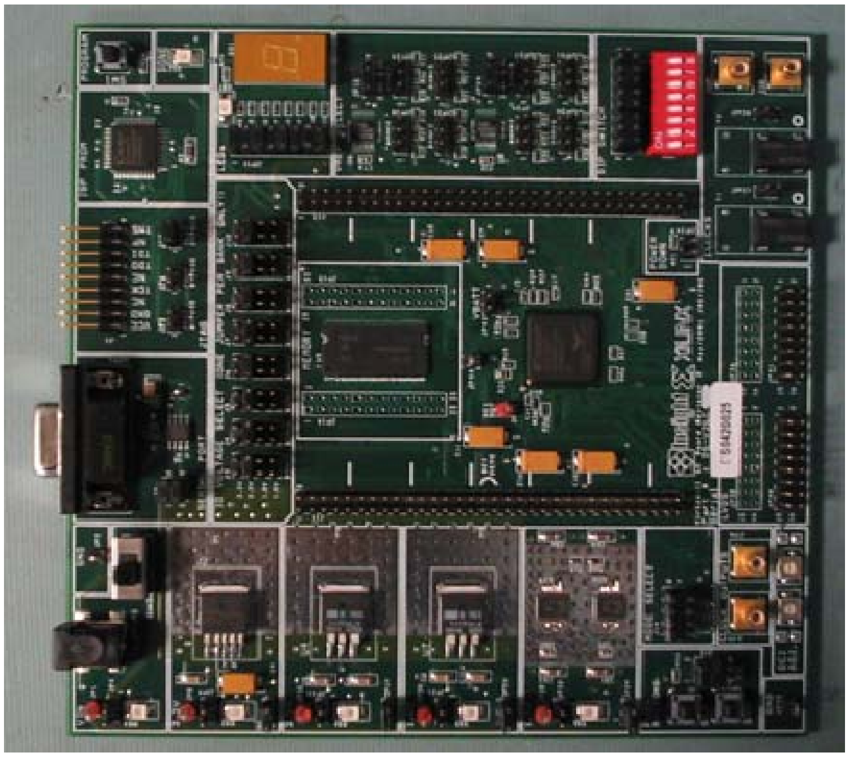


Figure 17. Virtex-II Development Kit.

B. CUSTOMIZED INTERFACE BOARD

The interface board was specifically designed with I/O ports, an A to D converter, and four voltage level shifters to interact with the Virtex II board configuration used in this thesis and to provide physical connections between the FPGAs and the other hardware devices in the design. The interface boards connected directly to the pins of the FPGAs. Jumper connections were also connected between the two boards to send the 5V supply from the interface to the Virtex II boards. The interface board connected to the Virtex II development kit with the 5V supply connected is shown in Figure 18. The layout of the interface board that was created using the PCB123 design software package is shown in Appendix C.

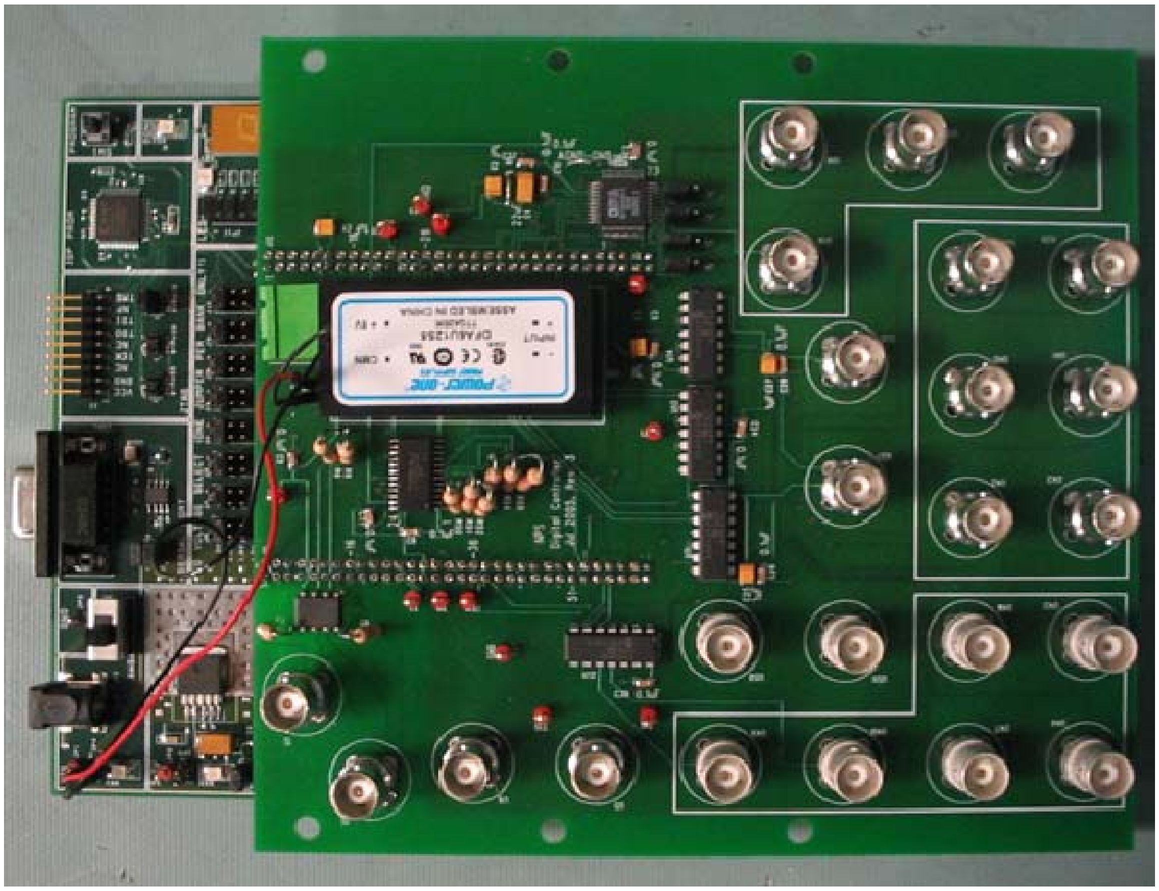


Figure 18. Customized Interface Board Connected to the FPGA.

Six BNC connections on the primary interface board were used to send out the gate signals from the selected controller to the six step three phase rectifier. Another BNC connector on the primary board was used to send out the fault signal to a BNC port on the secondary controller. Two more BNC connections on the primary board were used to send the theta synchronization signal and the serialized initial conditions out to the secondary board. That data was then sent into the secondary board through two resistor inputs. On the secondary interface board three BNC connections were used to send the gate signals of the secondary controller out to the primary board. Those signals were then sent into three input resistor connections on the primary board in order to be read by the switching unit in the primary FPGA.

The four feedback values from the VSI were sent into four additional BNC ports on each board that connected to the A to D converter to complete the closed control loop for each controller. Each controller also had a BNC that was used as an emergency manual shut off switch. The primary and secondary boards connected together are shown in Figure 19. This figure also shows the XILINX parallel cable connected to the primary board that was used to load the software through the JTAG port on the FPGA directly from the computer. The specific I/O ports and their corresponding FPGA pin connections in the software are covered in detail in the following chapters.

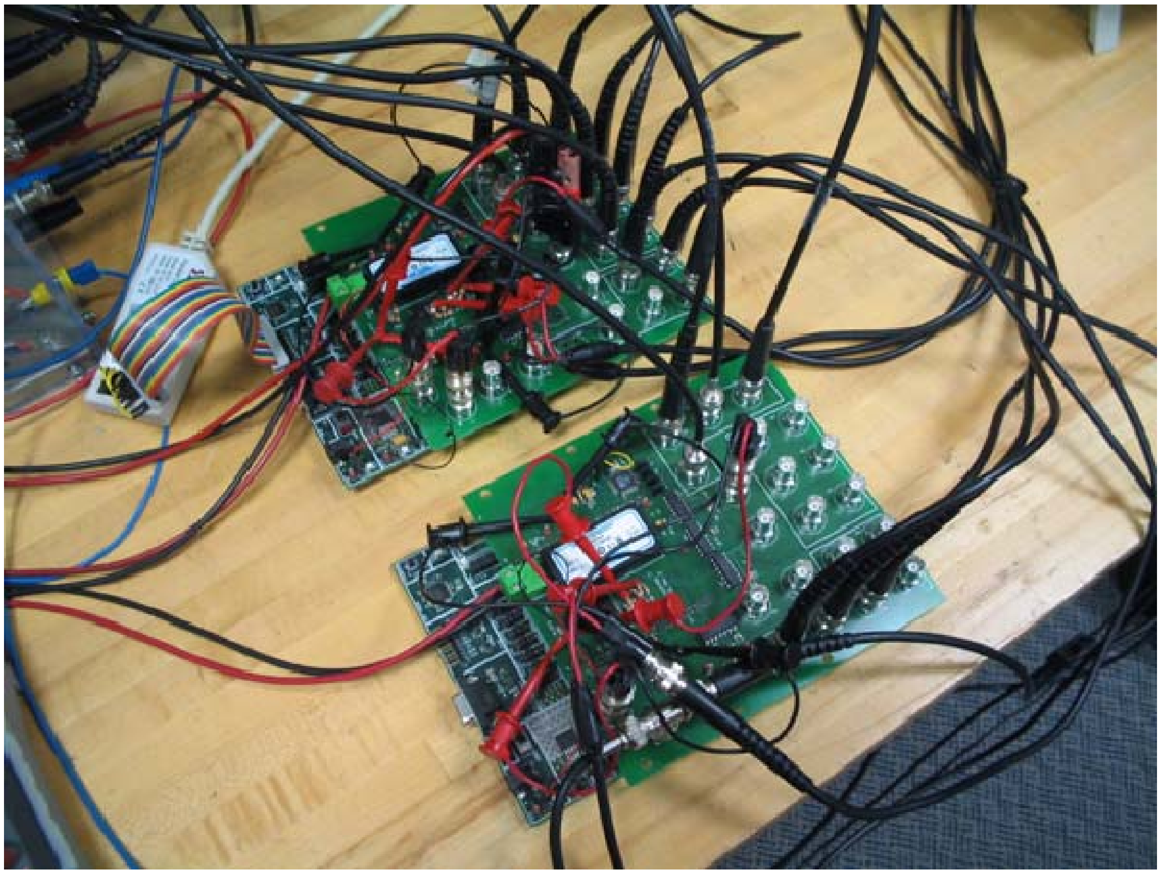


Figure 19. The Primary and Secondary Controller Boards Connected Together.

C. THREE PHASE RECTIFIER

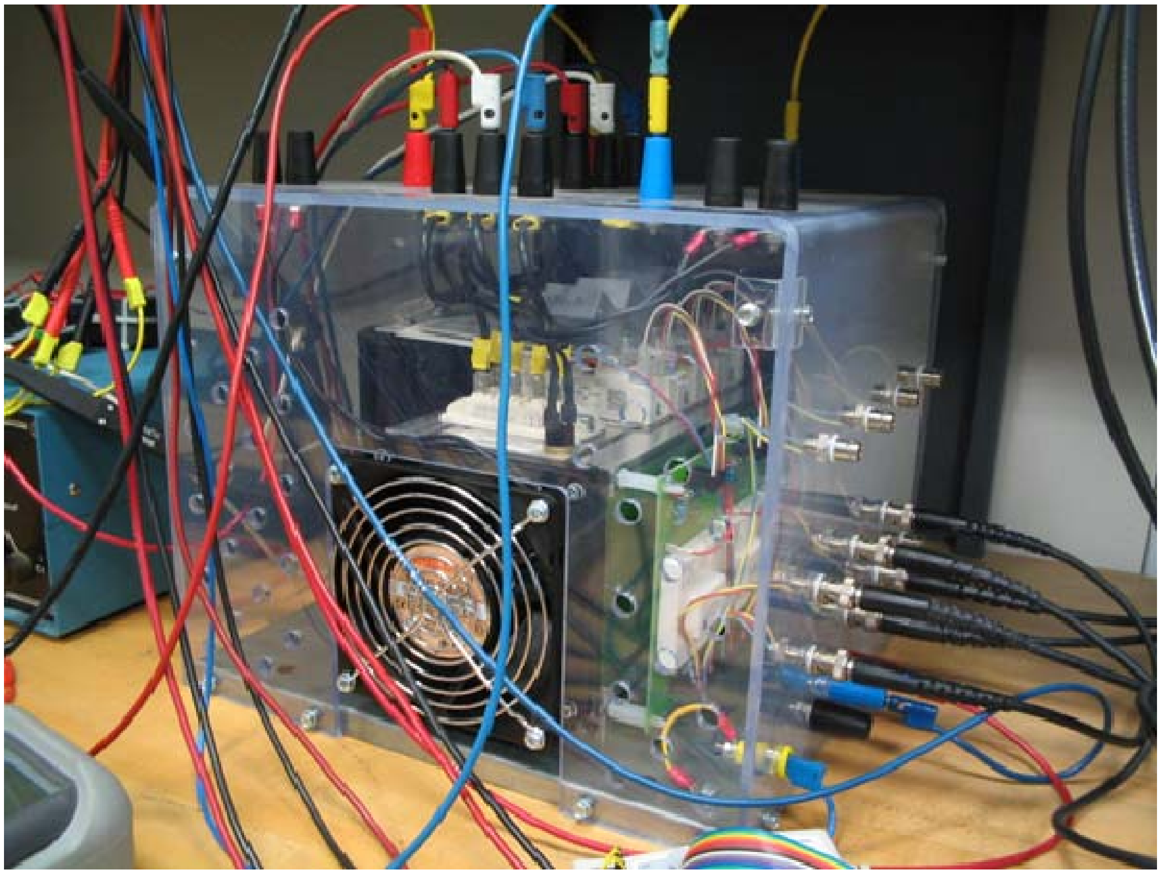


Figure 20. Three Phase Rectifier plus Inverter with Brake Chopper from SEMIKRON.

The SEMITEACH-IGBT used as the six step three phase inverter for the hardware configuration in this thesis is shown in Figure 20. The SEMITEACH is a multi-function IGBT converter with a brake chopper/rectifier. It is built with a transparent casing that allowed the operator to view the internal components. It also had several safety features which made it an ideal piece of equipment to be used in the laboratory environment [12]. Further technical specification for the SEMITEACH-IGBT is listed in Appendix D. The six gate signal outputs from the primary controller were sent into the six BNC connectors on the side of the SEMITEACH box that corresponded to the positive and negative gates for the three phases of the rectifier. A three phase AC power supply was applied to box through the three banana connectors on top of the box centered toward the front. Finally the three banana connectors centered in the top of the box produced the output of the three phase rectifier that was sent to the LC filter and on to the resistive load.

D. LOAD

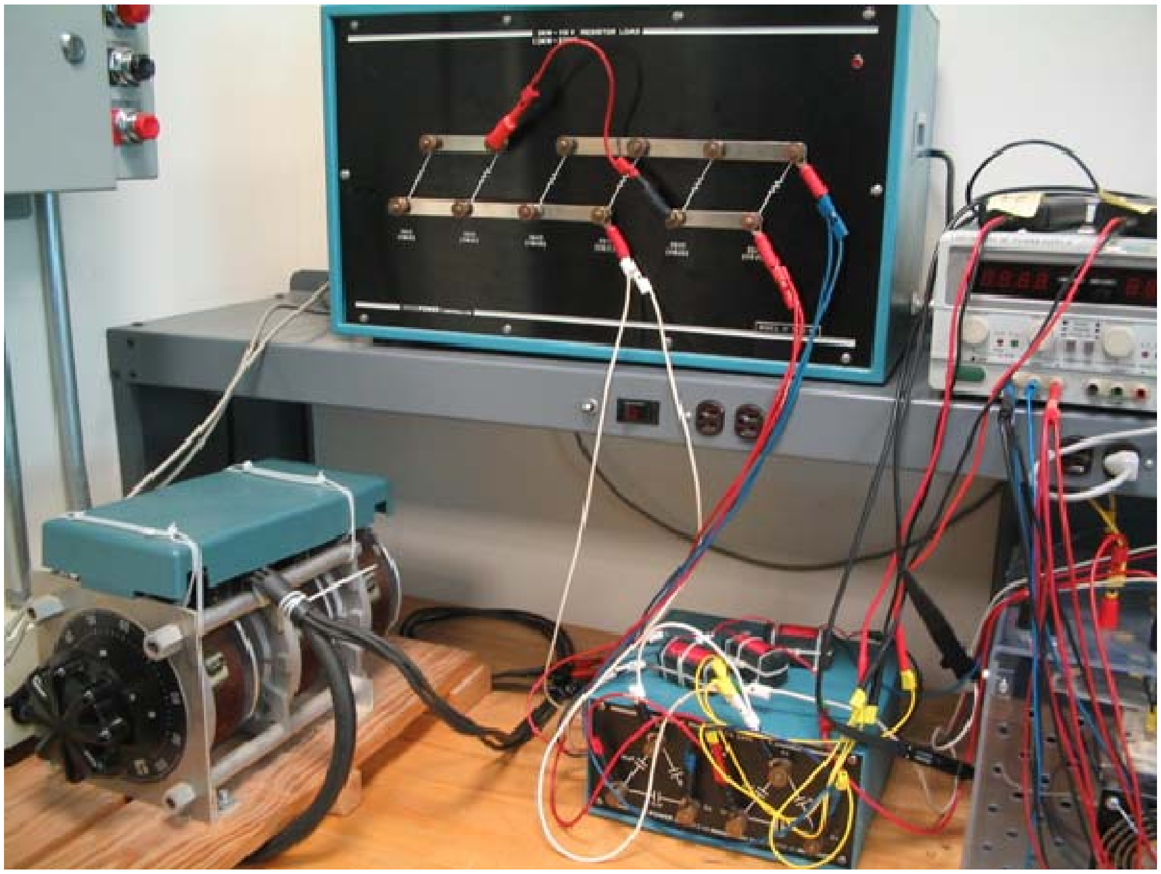


Figure 21. LC Filter and Resistive Load Setup.

The output of the three phase rectifier was sent out to three inductors connected to a delta configuration of capacitors to produce an LC filter. The value of the filter capacitors and the load resistors were adjusted to be equal to the computer simulated values. The inductors and capacitors of the LC filter had a value of $350\,\mu\text{H}$ and $20\,\mu\text{F}$ respectively. The LC filter limited the current and voltage in the time domain to produce a low pass filter that filtered out the modulation energy of the output and allowed the 100 Hz sine wave through before going into the load. Three $60\,\Omega$ resistors in a delta configuration were then used for the VSI load and the line to line voltage output across the resistors was measured for the experimental results. The LC filter and resistive load configuration in the hardware are shown in Figure 21.

E. HARDAWARE SETUP

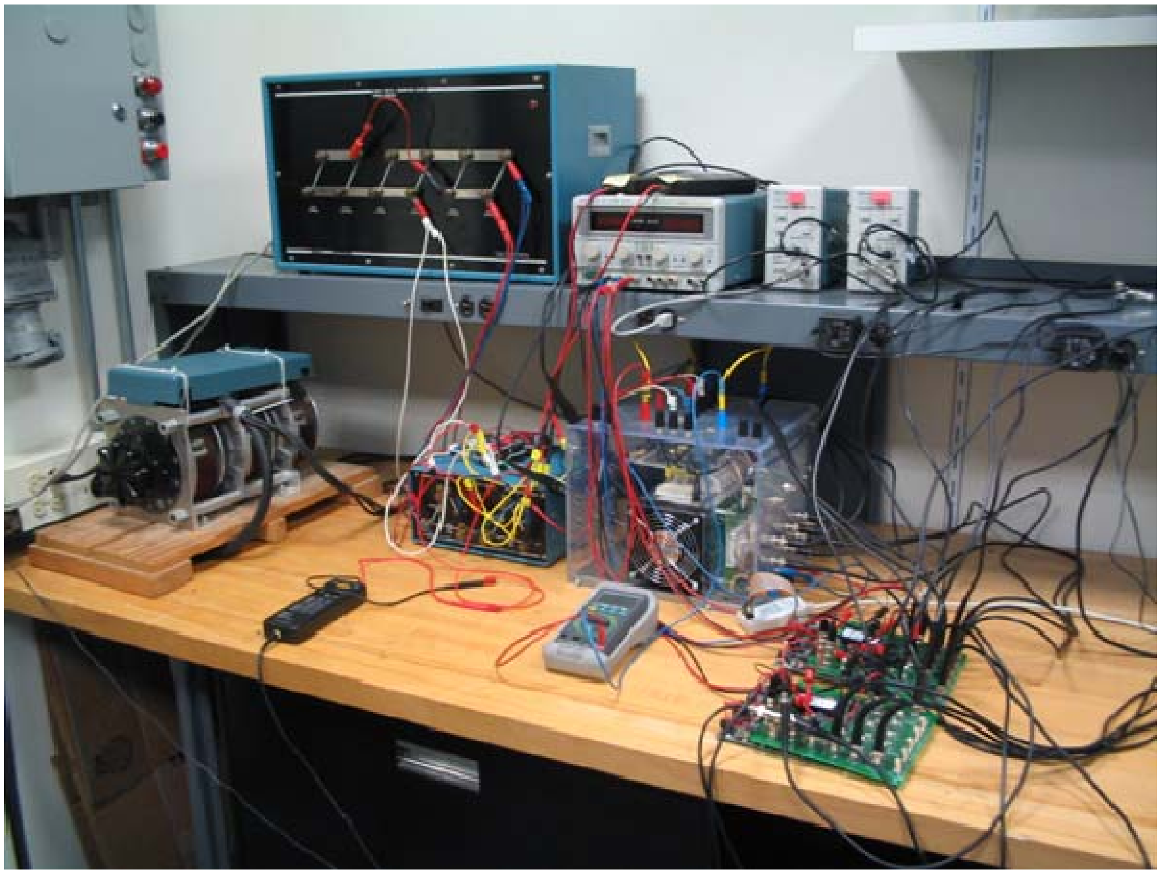


Figure 22. The Complete Hardware Design.

The components described in the preceding sections were finally implemented in the hardware design shown in Figure 22. This picture of the overall hardware design shows the hardware components described in the previous sections as well as the AC power supply for the three phase rectifier block, the DC power supply for the interface board, and the voltage and current probes used to provide the feedback signals for the closed loop design. Using this hardware configuration designed around the two FPGAs provided the ability to develop and test multiple software solutions for the redundant architecture without making any major changes to the hardware components. This ability saved both time and resources throughout the research and design process.

F. CHAPTER SUMMARY

This chapter described each of the hardware components used in the design to collect experimental data. The ability to physically separate the primary and the secondary controllers was vital to the experiments in order to ensure true redundancy was maintained in the design. Without physically separating the two controllers it would not have been possible to adequately demonstrate the data transfer necessary to eliminate the disturbance of the VSI output during the switching event. The next chapter describes the software design that was developed to achieve the first objective of the research which was to enable the VSI to switch from a primary controller to a secondary controller when a fault was detected. Each aspect of the redundant design is identified along with the FPGA and interface board pins assigned to the inputs and outputs of the controllers. The simulated and experimental results are then presented to confirm the design operated properly.

IV. INDEPENDENT REDUNDANT CONTROLLER ARCHITECTURE

This chapter discusses the original redundant controller design developed to achieve the first objective of this research to have a VSI with an independently operating redundant controller that would sense a fault in the primary controller and switch. The software used to create the redundant controller architecture consisted of a primary and a secondary controller block, an A to D converter block that read the feedback from the system, and a switching unit subsystem that switched between controllers when a fault was sensed. SIMULINK blocks were used to simulate the behavior of the three phase rectifier, LC filter, and resistive load in order to produce predictive simulation results prior to loading the software on the FPGAs. Finally, a Chipscope interface block was added to the software design which was able to set the fault signal during the computer simulations. The purpose for this additional software was to allow the operator to trigger the fault and shut off switches through the computer during the experimental tests rather than using physical switches.

In order to design the VSI with a redundant controller architecture the basic controller design discussed in Chapter II was given several modifications to enable it to sense input data as well as pass output data to the other elements of the design. The ability of the primary controller to communicate with the secondary controller and the switching unit were the first steps that had to be achieved. In the initial stages of the research one design was developed for both controllers which made it possible for the same code to be loaded on each board. However, as the design evolved to incorporate the passing of more data between the controllers it was not possible to keep the software exactly the same, and two separate configurations had to be developed.

The simulation software for the basic redundant controller design, which included the XILINX blocks that generated the code for the FPGA and the SIMULINK blocks that simulated the behavior of the external elements in the hardware, is shown in Figure 23. This configuration was used to collect simulated results in order to predict how the redundant controller architecture would behave in the real world system. The results obtained from this configuration gave reasonable estimates of the actual system's behavior and provided a high degree of confidence and predictability when moving to the real world experiments.

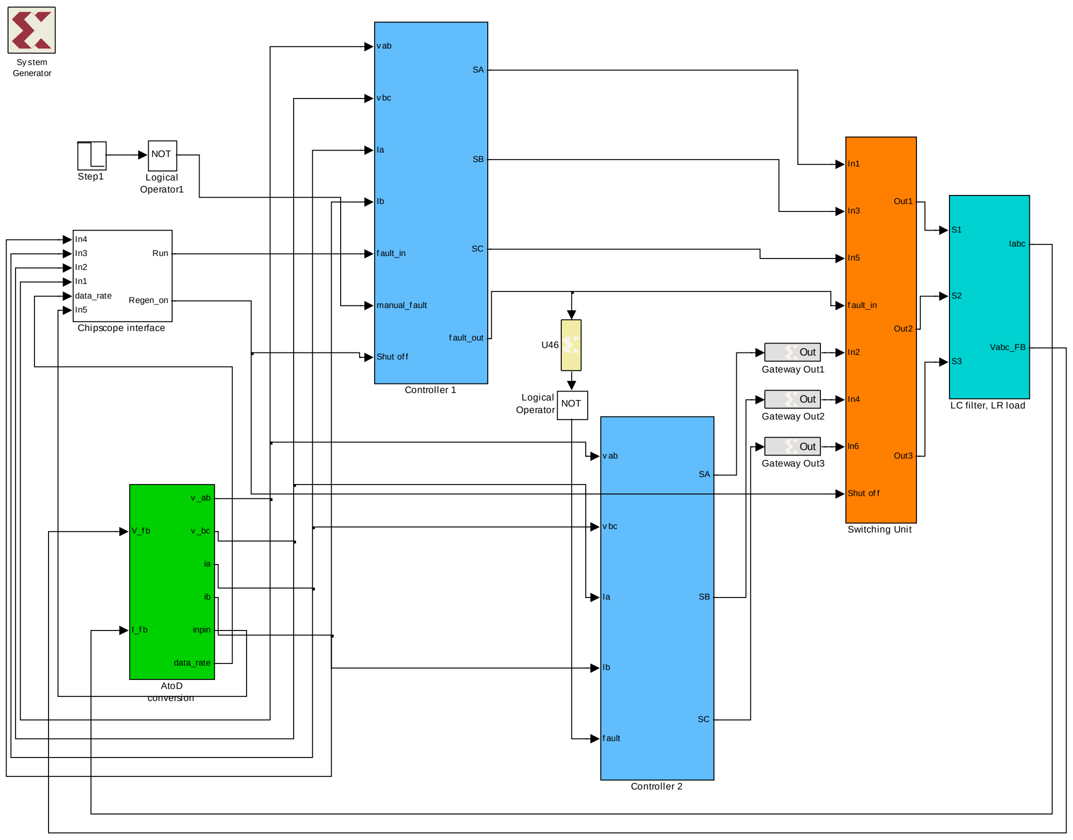


Figure 23. XILINX Model of the Redundant Controller System with Both Controllers Operating Independently.

The following sections address the issues considered during the initial stages of the design process and how the basic software components discussed in Chapter II were modified for this research. They also describe the new components that were developed and how all of the software elements interacted to create the redundant controller architecture.

A. FAULT SIGNAL

Although fault detection was an element of this research, it was not necessary to design the system to detect multiple types of specific faults in the controller in order to measure the disturbance of the output during the switching event. Therefore, the software was designed to simply read a high/low fault signal in the primary controller. The fault signal had the same input and output port assignments on both boards, and the input port on the interface card that read the fault signal was designed as an inverter. Therefore, when value of the fault signal is mentioned in this thesis it refers to the value of the signal being sent into the primary FPGA which is the inverted value of the manual switch signal going into the interface card.

The input port for the fault signal on the primary board was set up as a physical switch that could be used manually, and the input pin on the secondary interface card simply read the output from the primary interface card and inverted it before sending the signal into the secondary FPGA. An emergency shut off switch was designed into both boards to provide an extra level of protection for the equipment. The shut off switch was routed to the same BNC port on both interface cards. On the primary controller board the manual shut off switch was designed to turn off all of the signals being sent to the switches in the switching unit, preventing any gate signals from going into the three phase rectifier. The manual shut off switch on the primary controller board was also designed to send a signal that would reset the four PI accumulators of the primary controller. The manual shut off switch on the secondary controller board was designed to independently reset the four PI accumulators of the secondary controller. The physical input and output pins used for the fault signal and manual shut off switch on both boards are listed in Table 1.

Fault Signals	Vertex II (FPGA Pins)	Interface (BNC Ports)
Input	C4	U2
Output	T5	U46
Emergency Shut Off	D1	U1

Table 1. Fault Signal and Shut Off Switch Ports.

Additional XILINX blocks were included to enable the design to use Chipscope to perform the switching event. This addition was used to help reduce the risk of inadvertently damaging the boards by using the physical switches when collecting the experimental data. The software used for the Chipscope switch was the same software described in Chapter II, which provided the ability to use switches and to take internal readings of the system without using additional FPGA pins. Due to the fact that Chipscope could only be used through the XILINX parallel cable, the internal switches in the software could only be used on one board at a time. Therefore, the switching event was controlled through the primary controller board during the experimental testing.

1. Primary Controller

The fault signal on the primary controller could either be detected by the signal on the manual input switch or by the internal switch created by the Chipscope software. The fault signal was sent into the primary controller’s four PI control blocks, the switching unit, and the secondary controller’s four PI control blocks. When the fault signal was low the accumulators in the primary controller were enabled and the switching unit selected the primary controller’s gate signals to be sent into the three phase rectifier. The fault signal was also designed to be sent from the primary interface card to the secondary interface card, via a BNC cable, where it would be inverted to a high signal and sent into the secondary FPGA. Therefore, the system would continue to run with the primary controller until the fault signal in the primary FPGA went high indicating a failure of the primary controller.

2. Secondary Controller

Prior to a failure of the primary controller, the inverted fault signal going into the secondary FPGA kept the accumulators of the four PI controllers set to zero. This ensured the secondary controller would not start at some arbitrary value when it took over. When a fault was detected in the primary controller, the secondary PI controller's accumulators were enabled, and the switching unit selected the gate signals of the secondary controller to be sent into the three phase rectifier. The output pins of the three gate signals that were sent into the switching unit from the secondary controller are listed in Table 2.

Secondary Gate Signals	**Vertex II (FPGA Pins)**	**Interface (BNC Ports)**
SA	N6	U47
SB	P6	U48
SC	P7	U49

Table 2. Gate Signal Outputs for the Secondary Controller.

The ability of the secondary controller to accurately receive the fault signal from the primary controller and pass the output gate signals back to a switching unit was the first design requirement in creating the redundant controller architecture. In order to achieve this goal, the next step was to design a switching unit that could receive the output data from both controllers and reliably switch from the primary to the secondary controller. The next section discusses the switching unit design and how it managed the outputs of the two controllers to complete the systems redundant architecture.

B. SWITCHING UNIT

The switching unit design that determined which controller would be used by the voltage source inverter is shown in Figure 24. The switching unit was designed to read the gate signals from both controllers, the fault signal from the primary controller and the emergency shut off signal. The negative value of each gate signal input was created making a total of six signals from the primary and six signals from the secondary controller. Each gate signal from the primary and secondary controller were then sent to an AND logic gate along with the emergency shut off signal. The subsequent six values of each controller were sent to six individual switches that were designed to switch from the primary to the secondary controller values when the fault signal went high. The switching block then sent out the positive and negative values of the selected controller to the selected FPGA pins.

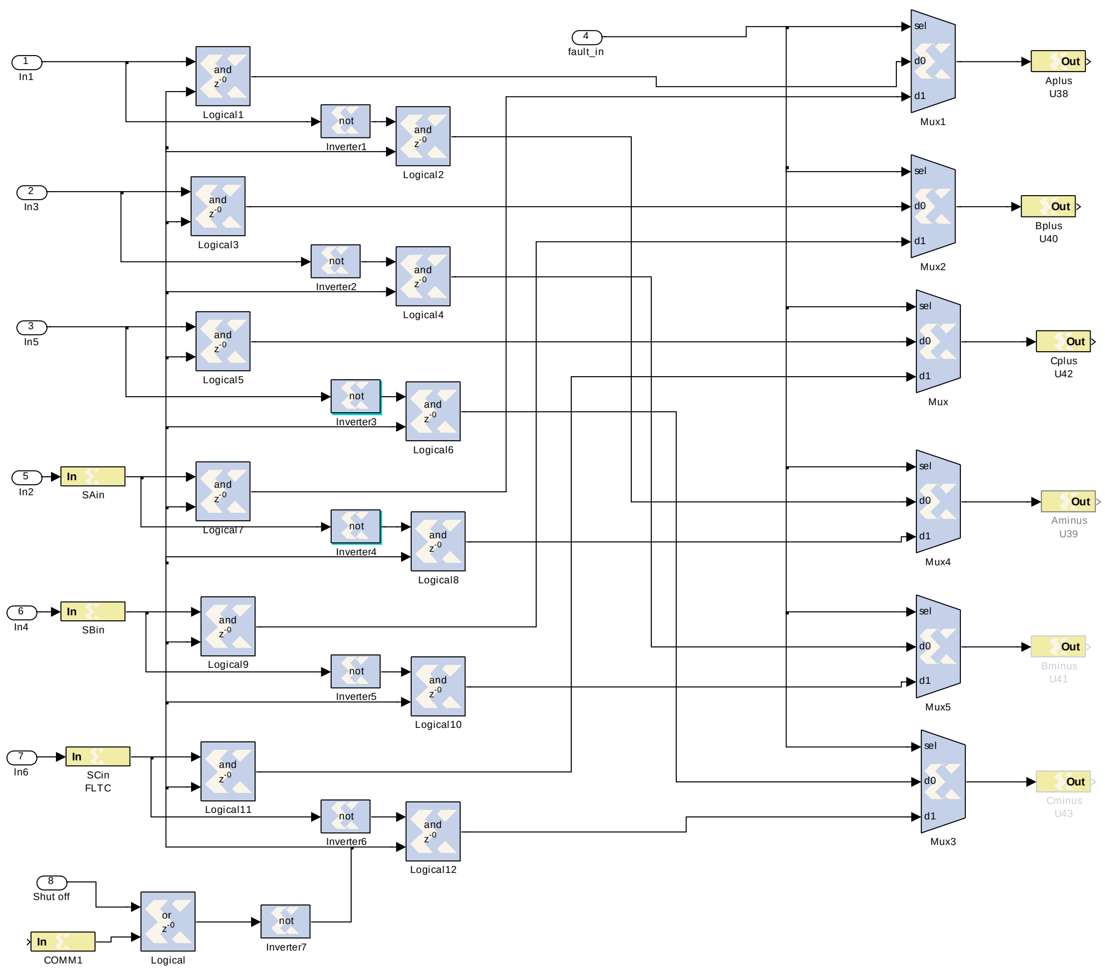


Figure 24. Switching Unit Software.

The values from the switching unit were sent from the FPGA pins to the BNC ports on the interface card which were then connected to the three phase rectifier. The gate signal values from the software and the corresponding FPGA and BNC connections are listed in Table 3.

Switching Unit Outputs	**FPGA Pins**	**BNC Ports**
A+	C16	U38
A-	D16	U39
B+	E13	U40
B-	H13	U41
C+	H14	U42
C-	H15	U43

Table 3. Gate Signal Ports on the Primary Board.

At this point the system was able to sense a fault in the primary controller and switch to a secondary controller without turning the system off which achieved the first research objective.

C. SOFTWARE ADDITIONS FOR EXPERIMENTAL TESTING

This section describes the software components that were not necessary for the redundant controller operations but were added to assist in the experimental measurements. A switching control subsystem was used in the experimental testing of the design to enable multiple measurements of a specific configuration to be properly compared. The switching events for the computer simulated results were able to be controlled by a simple step function that could be selected to switch at the same set time for multiple measurements. However, the switching event for the experimental results was triggered by an actual fault switch that was controlled by the operator. A switching event initiated at random by an operator was not capable of occurring at the same point of the output twice. Therefore, a second requirement, based on the value of theta was added to ensure that the fault signal that would trigger the switching event would always occur at the same point of the VSI output. This block of code was only necessary to collect the experimental data that need to be compared, and would not be included in any practical design.

The XILINX block design of the switching control software is shown in Figure 25. While the fault switch signal was low, the register block remained enabled which allow a low fault signal to be sent out. When the fault switch was activated by the operator, a high signal was sent into one input of the AND gate. The high fault switch also sent a low signal into one of the OR gate inputs. At this point the AND gate was still sending out a low signal, and the OR gate was still sending out a high signal to enable the register block. Therefore, the fault output was still low. The high fault output signal would not occur until a predetermined value of theta was also achieved. When both the fault switch and the theta value were selected, a high signal was sent out to the register block which passed the high signal. The high fault output signal was then sent back as a low signal into the second OR gate input which turned off the enable port and latched the high signal into the register. Latching the fault sign prevented the theta value from affecting the switching event any further.

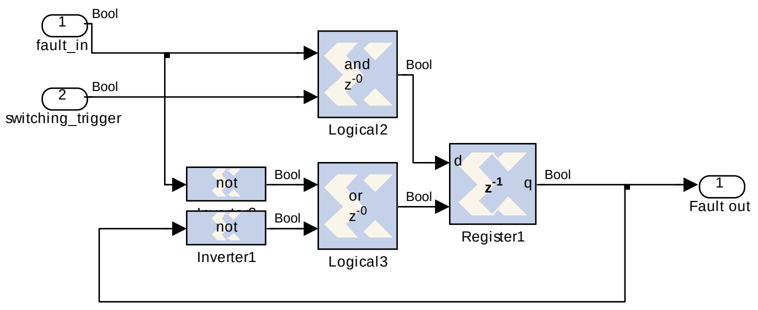


Figure 25. Switching Control Software for Experimental Results.

Another modification made to both the primary and the secondary controller at this stage of the research was a theta test point that would send a signal to an output pin on each board. The addition to the theta software block that produced the test signal that was collected during the experimental testing is shown in Figure 26. This test point was designed into the software to provide additional experimental data on the behavior of the theta values produced on each board. The test pin software was designed to send out a signal that would produce a rising edge when the value of theta reached $\frac{5}{3}\pi$ and a falling edge when the theta accumulator reached 2π and reset to 0.

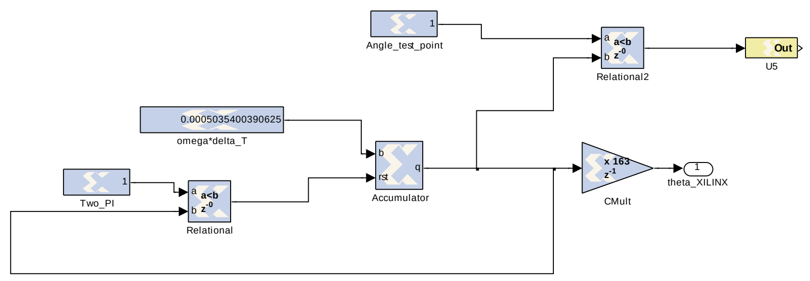


Figure 26. Theta Test Pin.

The FPGA pin used for the theta test output and its corresponding interface board connection for both controllers is listed in Table 4.

Theta Test Pins	**FPGA Pins**	**Interface Connection**
Test Output	H4	U5

Table 4. Theta Test Pins.

D. SIMULATED RESULTS

Three measurements of a single phase of the simulated output when the VSI switched between two independently operating controllers with no synchronization is shown in Figure 27. Only one phase is present on the graph in order to better see the disturbance produced by not having any synchronization between the primary and secondary controllers. Since both controllers were operating in a single computer program the theta offset in the three measurements had to be added to the secondary controller manually to try and accurately simulate two physically separated clocks counting at slightly different rates. The graph shows a period of about 0.08 seconds between the switching event and the secondary controller achieving steady state operations. It is important to note that this disturbance time could be longer or shorter depending on the gain values chosen for the system. The disturbance in the VSI output shown in the simulated results indicated the two main sources of the disturbance came from the phase shift due to the internal theta values in each controller and the transit time required for the secondary controller to achieve steady state after the switching event occurred.

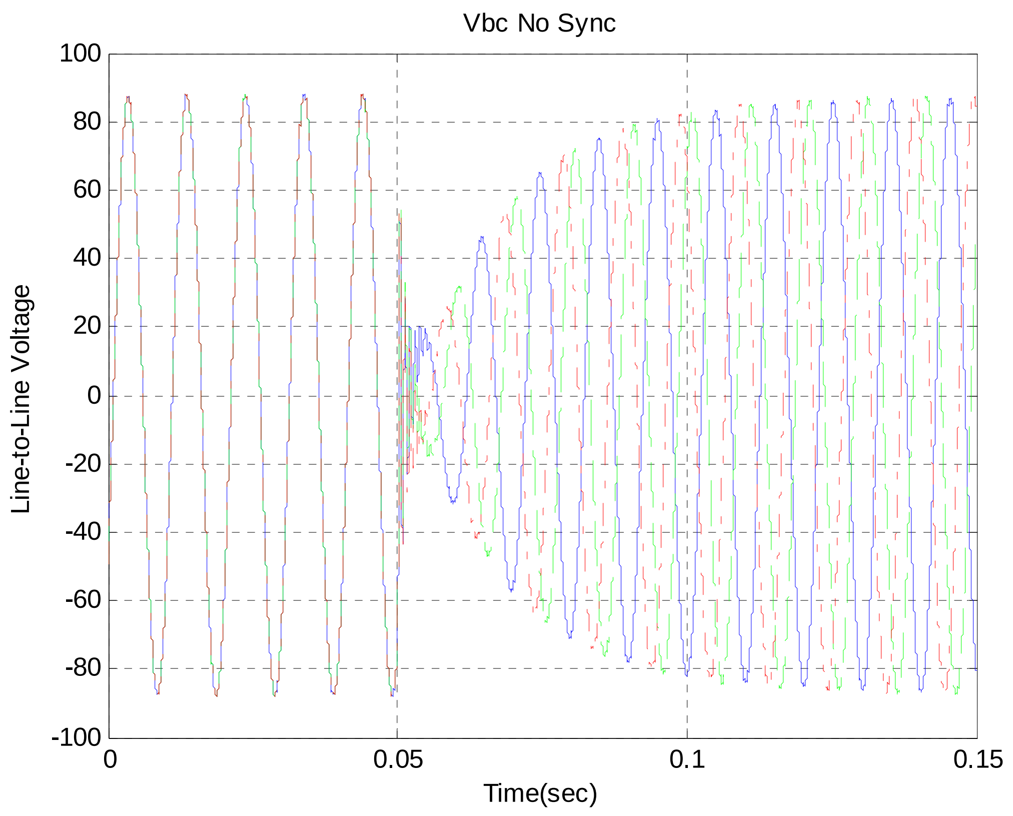


Figure 27. Three Simulated VSI Outputs with Random Theta Values for the Secondary Controller with the Switching Event at 0.05 seconds.

E. EXPERIMENTAL RESULTS

The experimental results of the redundant system at this stage of development are shown in Figure 28. The graph displays three separate measurements on top of each other which exhibit three separate, random phase shifts in the output of the VSI. The graph also shows the major disturbance in the amplitude of the output while the secondary controller’s accumulators achieved steady state. The behavior of the experimental VSI inverter output was very similar to the predicted behavior produced by the simulated outputs. The similarities of these two outputs provided a high level of confidence in the predictability of the computer simulation software used in this research.

Both the simulated and experimental results show a significant disturbance in the amplitude and phase of the output during the switching event. These results illustrated the need to develop the design further in order to produce an output that would meet military standards [1].

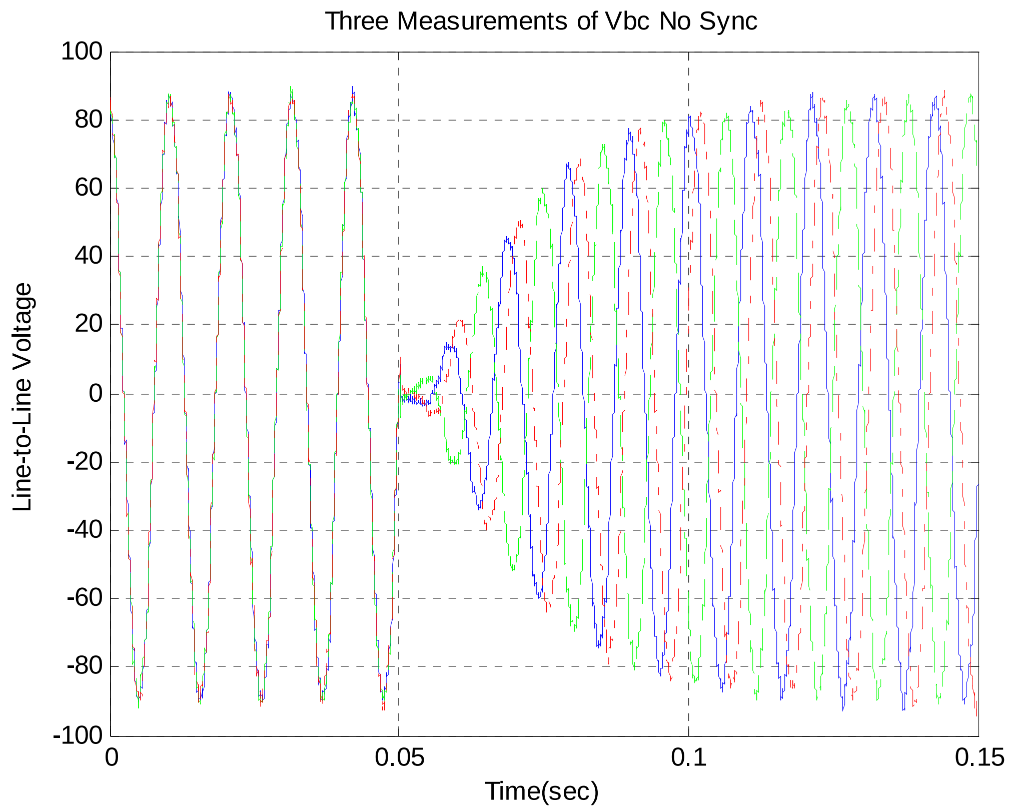


Figure 28. Three Single Phase Experimental VSI Output Measurements with No Synchronization of the Primary and Secondary Controllers' Theta Values and the Switching Event at 0.05 seconds.

Further confirmation that the phase shift in the output was due to the internal theta values of the two controllers drifting apart over time was provided by collecting experimental data from the theta test pins in the controllers. The outputs of the theta test pins on the primary and secondary boards at a random point of operation are shown in Figure 29. The solid blue and broken red graphs represent the primary and secondary controllers' theta values respectively. The rising edge of these graphs indicate when the theta value of each controller reached $\frac{5}{3}\pi$, and the falling edge indicates when the theta values reached 2π and reset. The clear difference in these experimental results helped to verify the computer simulations predictions that the phase shift during the switching event was being caused by the variation in the two independently operating theta values of the controllers.

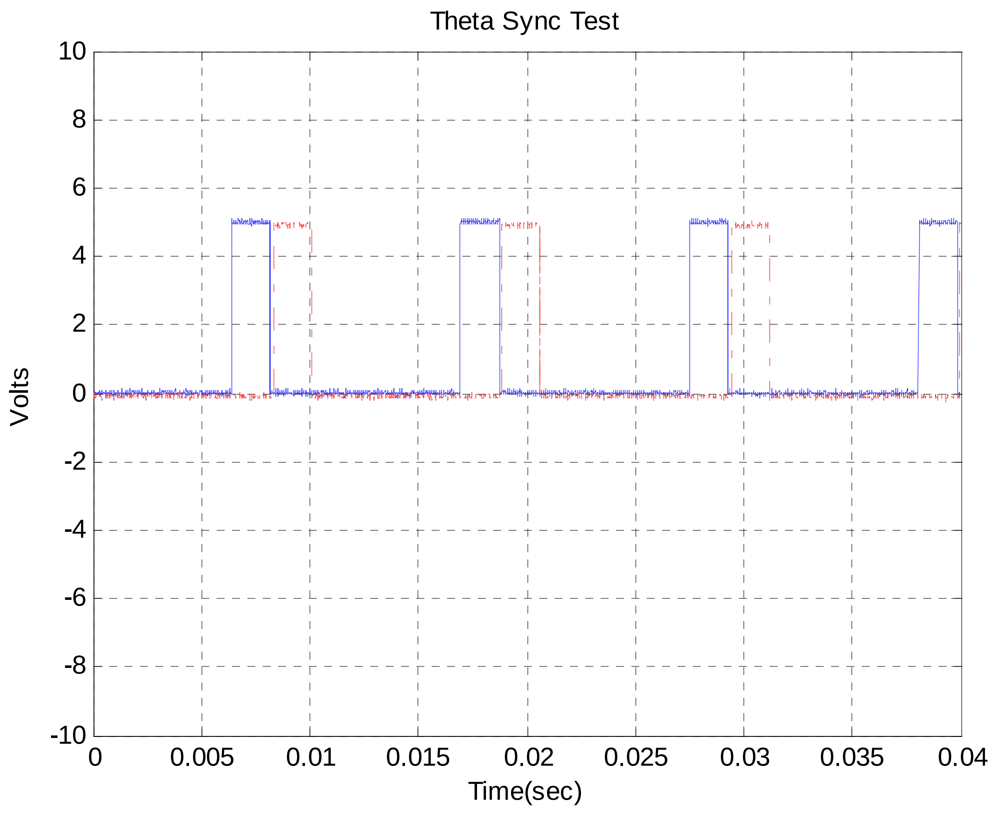


Figure 29. Theta Pulses of the Two Controllers without Synchronization.

F. CHAPTER SUMMARY

This chapter provided a break down of the software design modifications made to both the primary and secondary controllers in order to produce a VSI with operating standby redundant controller architecture were both controllers operated independently of each other. The simulated behavior of the redundant controller architecture presented in this chapter made it possible to develop the software design more thoroughly prior to implementing it in the hardware. The ability to see the computer generated behavior of the entire system was extremely beneficial in troubleshooting the software and guiding the design development. The computer simulations of this design also provided information with which to compare the experimental measurements of the software when it was loaded in the FPGAs. The next chapter discusses the design issues involved in eliminating the random phase shift of the VSI output during the switching event, and the solution that was implemented to achieve the second objective of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

V. PHASE SYNCHRONIZED REDUNDANT ARCHITECTURE

This chapter presents the software solution developed to eliminate the random phase shift in the VSI output that was produced during the switching event with two independently operating controllers. The modifications to the software of the primary and secondary controllers in order to synchronize the theta values are explained. The computer simulations of the VSI output are then presented which predict the behavior of the design. Finally the experimental results are displayed to confirm the real world operation of the new configuration.

A. THETA SYNCHRONIZATIONS

Once the system was able to effectively switch from the primary to the secondary controller it was necessary to attempt to minimize the disturbance of the system output during the switching event. Theta synchronization was the first step in controlling the output of the secondary controller when the fault occurred. Since the controllers were loaded on separate boards to help ensure greater reliability of the system during a failure of the primary controller, the theta values of each controller would slowly drift apart over time without some mechanism to keep them aligned. This difference in the internal theta value of the two controllers would cause the output of the VSI to have a random phase shift when the system switched from the primary to the secondary controller. Another issue that had to be considered was the fact that any information passed from one FPGA board to another would require the use of one output pin per bit of data.

Since the theta value for each controller was designed to count from 0 to 2π and then reset the accumulator back to zero, the theta values could remain synchronized by simply having a single pulse that would reset both theta accumulators at the same time. It was also necessary to ensure that the synchronization between the primary and secondary values of theta would not continue after a fault was detected in order to maintain true redundancy. This single pulse method was considered the best solution because it only required a single bit to pass the information needed to keep the theta value of both controllers synchronized.

1. Primary Controller

The modification that was made in the theta block of the primary controller enabled it to pass the reset signal to the secondary controller before the fault was detected and stop passing the signal after the detection of the fault. This modification is shown in Figure 30. The reset pulse is sent to an AND logic gate with the inverted fault signal. Prior to a failure, the inverted fault signal remains high and the AND gate will pass the reset signal to the secondary controller keeping it synchronized with the primary controller theta value. Once a failure in the primary controller is detected, the inverted fault signal will go low which will prevent the reset pulse from the primary controller from continuing to pass to the secondary controller's theta block.

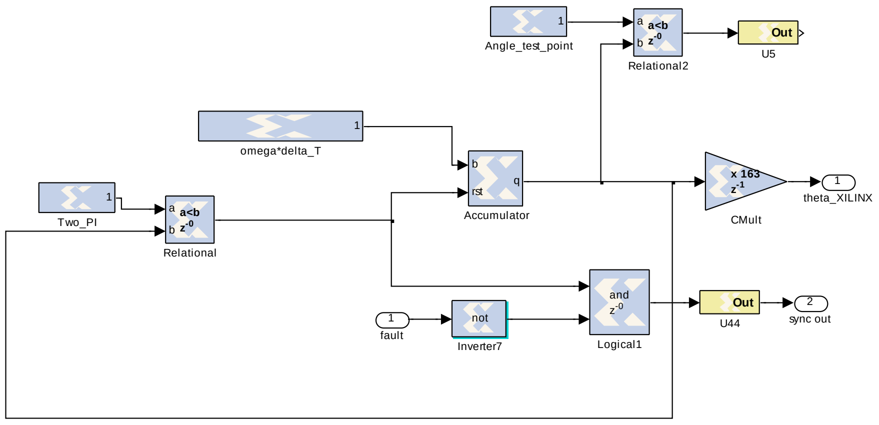


Figure 30. Theta Synchronization Software for Primary Controller.

2. Secondary Controller

The design modifications to the secondary controller's theta block are shown in Figure 31. The secondary controller was designed to accept the reset pulse from the primary controller as well as continue to operate on its own once the pulse from the primary controller stopped being sent. This was accomplished by bringing the reset signal from the primary controller into an OR logic gate with the secondary controllers reset signal. Since the synchronization pulse was delivered at the end of every 2π cycle, any difference in the theta values of the primary and secondary controllers during that time would have been negligible. Therefore the reset signals from the primary controller and the secondary controller would occur at the same time. The OR logic gate in the design allows the theta software to continue to reset after the synchronization signal is discontinued due to the detection of a fault in the primary controller.

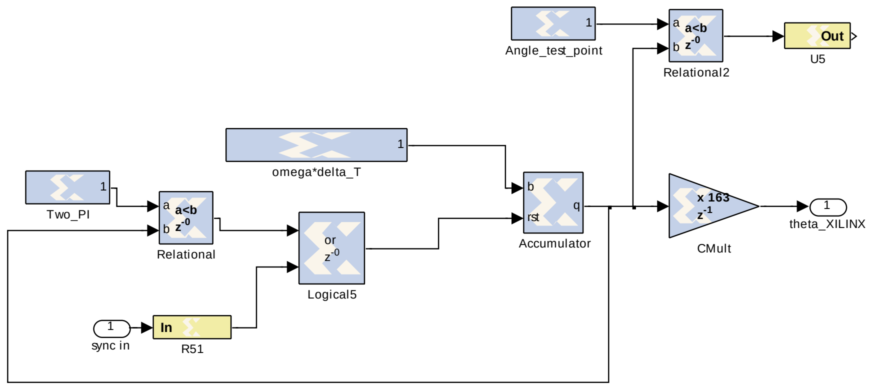


Figure 31. Theta Synchronization Software for Secondary Controller.

The input and output pins used to synchronize the theta values of the primary and secondary controllers are listed in Table 5. The FPGA pin on the primary board sent the pulse signal to the primary interface board's BNC connection listed in the table. The primary interface board then sent the signal out to the secondary interface board's BNC connection, which was then sent into the FPGA input pin on the secondary board listed in the table.

Synchronization pins	**FPGA Pins**	**Interface Connection**
Primary output	L5	U44
Secondary input	K3	R51

Table 5. Theta Synchronization Connections.

B. SIMULATED RESULTS

The theta offsets that were added to the secondary controller for the computer simulations in Chapter IV were taken out, which automatically synchronized the theta values of the two controllers in the computer simulation. The simulated result of what the VSI output should look like without any phase shift during the switching event is shown in Figure 32. The fact that the theta values of the two controllers were identical because the simulation was run on one computer made it difficult to determine if the theta synchronization software was working. However, the simulated result still provided an accurate output with which to compare the experimental results.

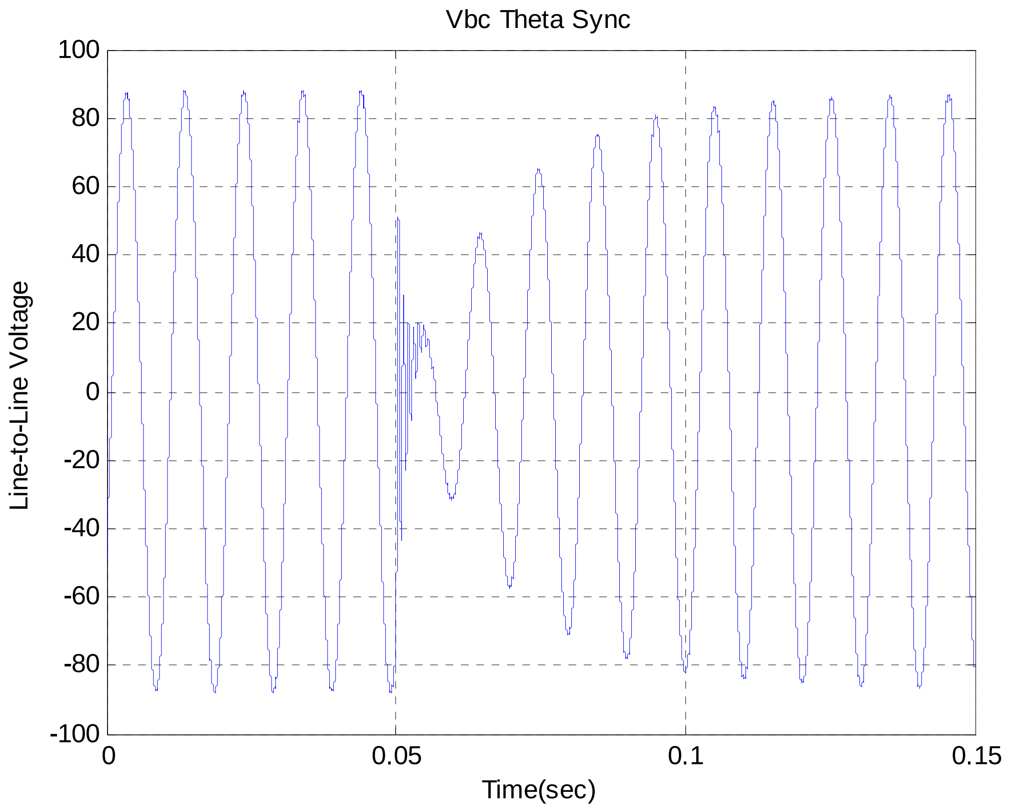


Figure 32. Simulated VSI Output with the Theta Values Synchronized and the Switching Event at 0.05 seconds.

C. EXPERIMENTAL RESULTS

The experimental tests in this section confirmed that the theta synchronization software added to the design eliminated the phase shift to the VSI output during the switching event. The three experimental results of the VSI output for the redundant controller system with the theta synchronization pulse being sent from the primary to the secondary board are shown in Figure 33. The switching control software was also implemented for this experimental test in order to properly show the ability of the VSI to maintain the same output phase during three separate failures of the primary controller.

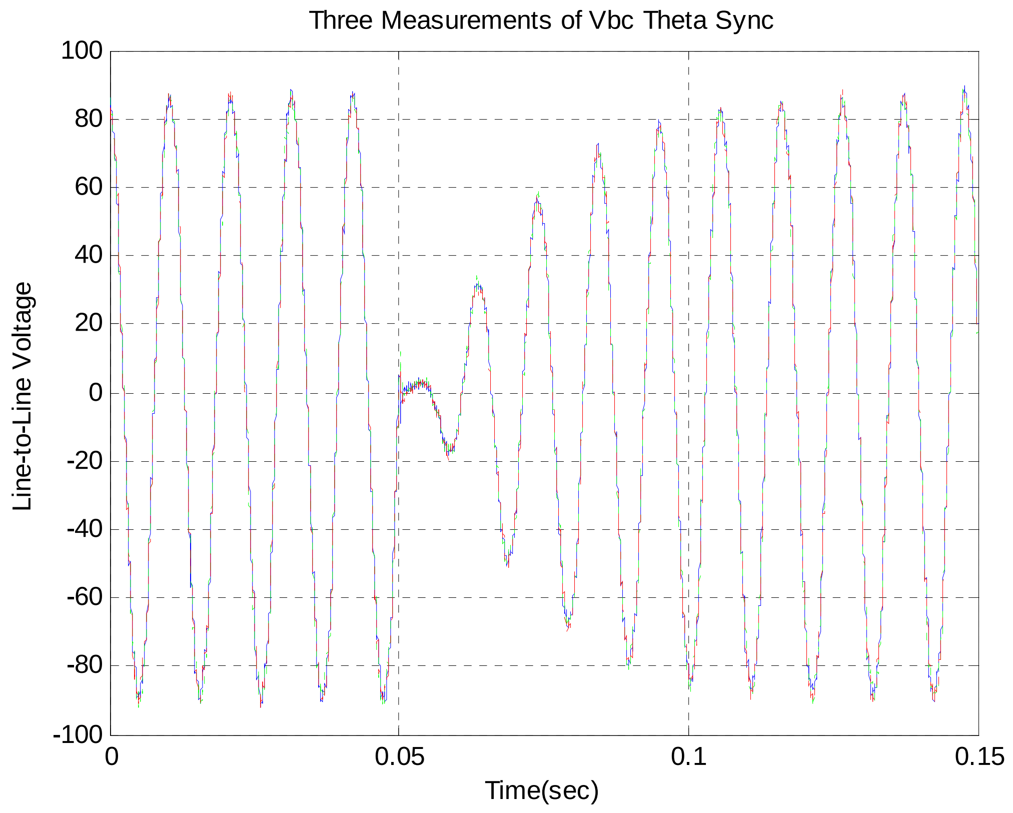


Figure 33. Three Single Phase Experimental VSI Output Measurements with Theta Synchronized and the Switching Event at 0.05 seconds.

The behavior of all three experimental VSI inverter outputs was very similar to the predicted behavior produced by the computer simulation. Although the disturbance in the amplitude was still present during the switching event, the random phase shift element of the disturbance had been removed by adding the theta synchronization pulse. Although this still did not meet the military standards for voltage disturbance in a power system [1], the ability to predict the disturbance in the output was a significant improvement over the previous design presented in Chapter IV.

A reevaluation of the theta test pins with the theta synchronization software implemented showed that the theta values of the two controllers remained identical prior to the switching event. The output of the theta test pins prior to the primary controller failing is shown in Figure 34. The solid blue line and the broken red line indicate the primary and secondary controller respectively. The figure shows that the rising edge, which indicates when the theta values are $\frac{5}{3}\pi$, and the falling edge, which indicates when the theta values are 2π were identical in both controllers before the primary controller failed.

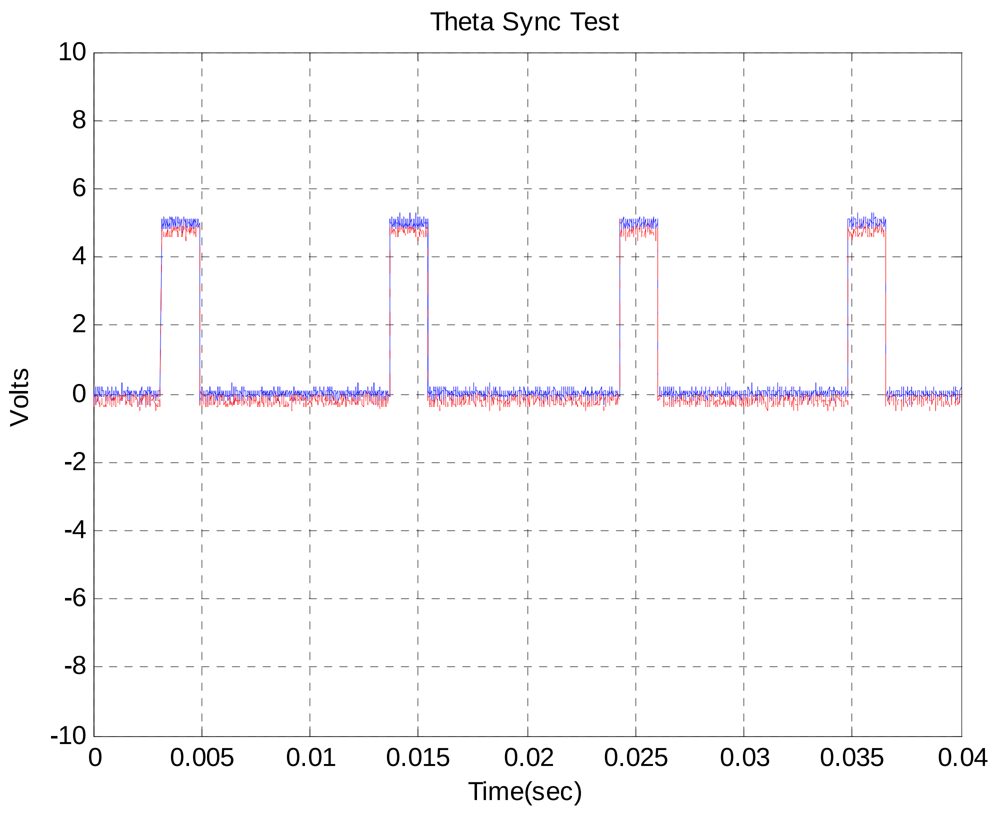


Figure 34. Theta Pulses of the Two Controllers with Synchronization Prior to the Fault.

The output of the theta test pins twenty seconds after the primary controller failed is shown in Figure 35. It is apparent from the two graphs that the theta values ceased to be synchronized once the secondary controller took over, and the two values slowly began to drift apart after the fault occurred. This experimental result also confirmed that the feature in the software that helped to ensure that the secondary controller would not be potentially corrupted by continuing to be connected to the primary controller after the failure was functioning properly.

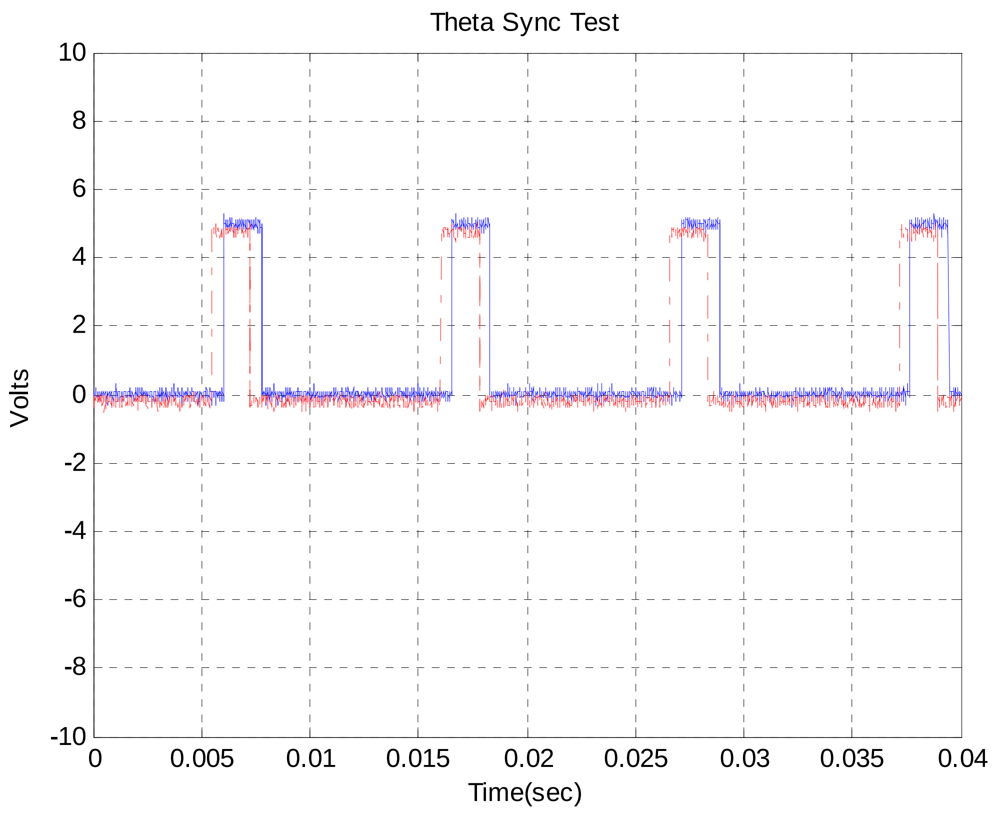


Figure 35. Theta Pulses of the Two Controllers with Synchronization Twenty Seconds After the Fault.

D. CHAPTER SUMMARY

This chapter discussed the software design modifications made to the theta software in the primary and secondary controllers to eliminate the random phase shift in the VSI output during the switching event. The simulated results of the design predicted the pulse from the primary controller's theta software would synchronize the theta values of the two controllers at the end of every 2π cycle. Multiple experimental measurements provided confirmation that the theta synchronization software was able to eliminate the phase shift in the output. Additional confirmation that the theta synchronization was discontinued after the fault was detected was also provided from the theta test pins located on both boards. This experimental information helped to confirm that the redundancy of the system was being maintained. The next chapter discusses the design issues involved in passing initial condition values to the four PI control blocks from the primary controller to the four PI control blocks of the secondary controller to eliminate the drop in the amplitude of the VSI output during the switching event.

VI. FULLY SYNCHRONIZED REDUNDANT ARCHITECTURE

The final goal of the research was to design the redundant controller architecture so that the secondary controller would come online at the same place the primary controller failed. The final XILINX model that achieved all of the objectives of this research is shown in Figure 36.

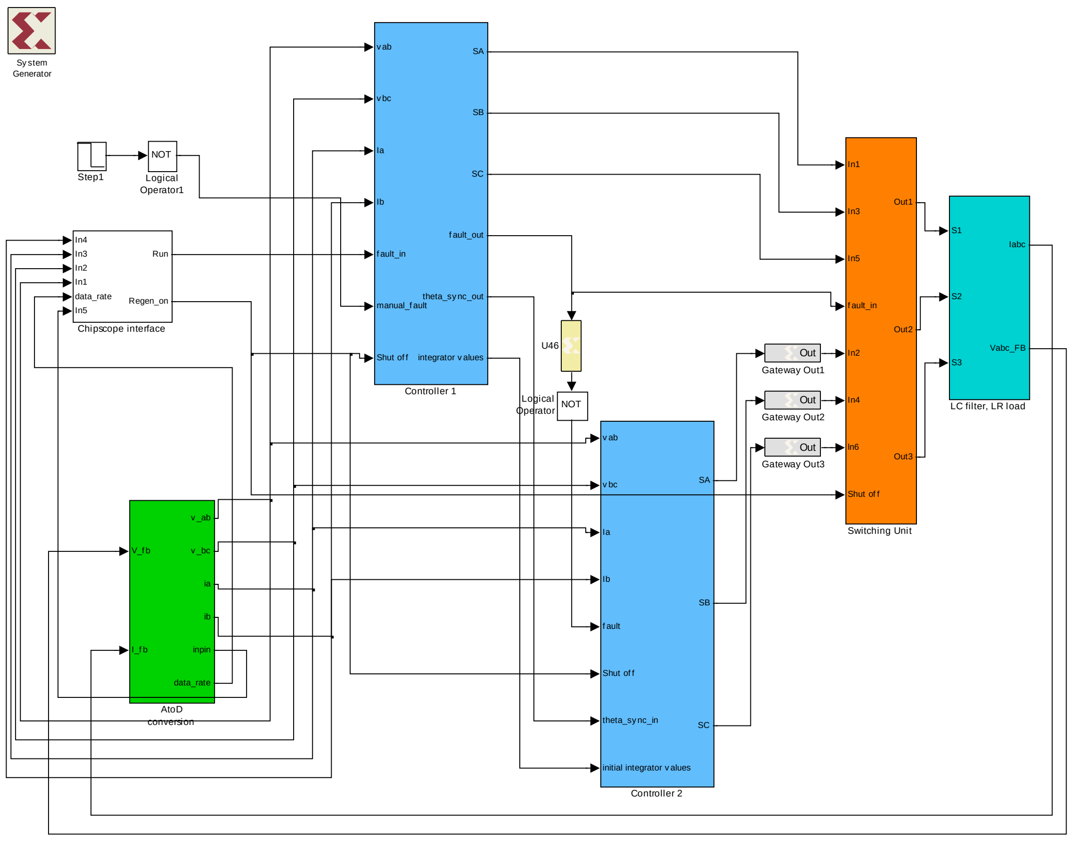


Figure 36. XILINX Model of the Redundant Controller Design with the Theta Synchronization and the Integrator Values of the Primary Controller Passed.

Until this point in the design it was necessary to keep the secondary controller's integrator values set to zero to prevent it from coming online at some random point and potentially damaging the VSI. However, keeping the integrator values at zero prior to sensing a fault in the system meant that the VSI output would have to start at zero and work its way back up to steady state. Although the loss of power would be brief, it would not be an acceptable design to meet military standards [1]. This problem could be solved by sending the four integrator values from the primary controller to the corresponding integrators in the secondary controller to be used as a starting point when it came online. However, the process of sending data between two physically independent FPGAs added an extra degree of difficulty to that approach. In order to send a binary value out of the FPGA there had to be one FPGA pin assigned for each bit of the value. Therefore, if four twelve bit words were to be sent, forty-eight output pins on the FPGA would need to be used to send each bit out. Since this would not have been a feasible solution given the resources used in this research, an alternate way of passing the data had to be developed.

A. INITIAL CONDITIONS TRANSFER SOFTWARE

The solution that was implemented to achieve the objective of passing the initial conditions from the primary to the secondary controller was to create additional code that would serialize each of the twelve bit integrator values from the primary controller and concatenate them into a string of forty-eight single bits. By doing this the data was able to be sent using a single output pin on the primary FPGA and a single input pin on the secondary FPGA. Additional code also had to be developed for the secondary controller to deserialize the string of data back into four separate values that could be sent to the appropriate integrators.

The serialization portion of the software was designed to take in four binary values of any size, convert them to twelve bit values with specified binary points, and reinterpret them into unsigned twelve bit values. The four unsigned twelve bit values were then sent into a XILINX Mcode block where MATLAB code selected the four values to be sent out to another Mcode block that serialized the bits into a Manchester format. The final serialized output consisted of a start bit, a forty-eight bit string, and a stop bit. This string was then sent into the deserializaion portion of the software were the bits were reassembled back into four individual values. The Manchester coding scheme used in the software interpreted a rising edge in the middle of a bit as a zero and a falling edge as a one. The serialization code is listed in Appendix F. Once the serial to parallel software was shown to work reliably in the software, the four PI blocks in both the primary and secondary controllers were modified to send and receive the data properly.

1. Primary Controller

The only change made to the primary controller was the addition of the serialization portion of the software. No modifications to the PI control blocks were needed to be made other than to send the integrator values to the serialization block. The primary controller sent out all four integrator values into the serialization portion of the software. The four values were serialized into a string of forty-eight bits and then sent out a single bit output gate.

2. Secondary Controller

The secondary controller received and decoded the serialized data from the primary controller using the deserialization portion of the software. The serialized bits were then broken back out into four twelve bit unsigned values. Finally the four values were reinterpreted into 2's complement values with the appropriate binary points. The appropriate integral value was then read by the corresponding PI control blocks through a register latch until a fault occurred. When the secondary controller sensed the fault signal, it took the last inputs sent to the four integrators and latched the values to be used as the initial conditions for the secondary controller. The modifications made to the V^e_q integrator are shown in Figure 37. The same design was used for all four integrators of the secondary controller to achieve the initial conditions transfer.

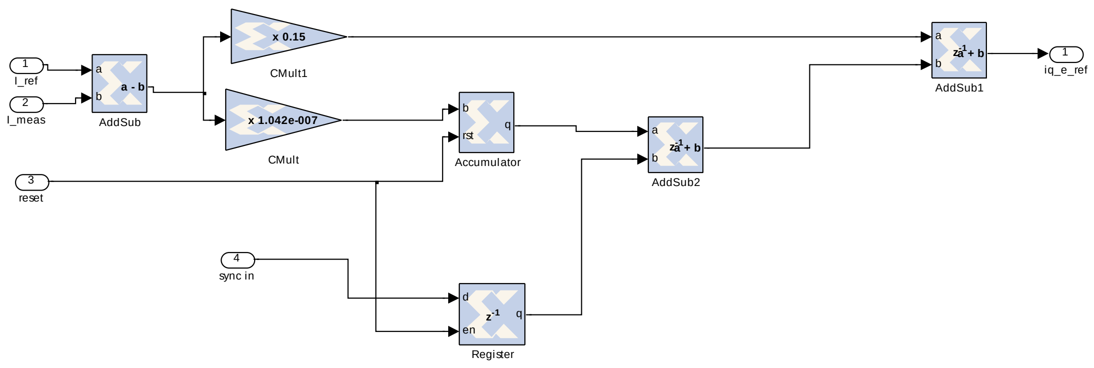


Figure 37. Initial Condition Configuration for the Secondary Integrators.

The output pin used to send out the serialized data from the primary controller and the input pin used to receive the data on the secondary controller are listed in Table 6.

Serialization pins	**FPGA Pins**	**Interface Connection**
Primary controller output	T6	U45
Secondary controller input	C8	R54

Table 6. Serialization Pins.

B. SIMULATED RESULTS

1. Simulated Test of Serialization Software

In order to test the code, computer simulations were conducted to demonstrate that the serialization software could take in four distinct twelve bit values, send out one bit at a time, and reassemble the bits back into the same four bit values. The Manchester output of the serialization portion of the software with the four constant input values of one, two, three, and four are shown in Figure 38. Each of the input constants were set as sixteen bit 2's complement values with binary point values of three. The serialization software then took each of the constant values and recast them as twelve bit values with varying binary points assigned to work in the actual software design. The binary points for the one through four values were eight, eight, two, and six respectively. The new twelve bit values were reinterpreted into unsigned twelve bit values with no binary point before being serialized and concatenated into a string of forty-eight bits. The four individual binary values and the subsequent forty-eight bit string are shown in equation (6.1).

$$\begin{aligned}&\text{Constant One} = 000100000000\\&\text{Constant Two} = 001000000000\\&\text{Constant Three} = 000000001100\\&\text{Constant Four} = 000100000000\\&\text{Serialized String} = 000100000000001000000000000000001100000100000000\end{aligned}\qquad(6.1)$$

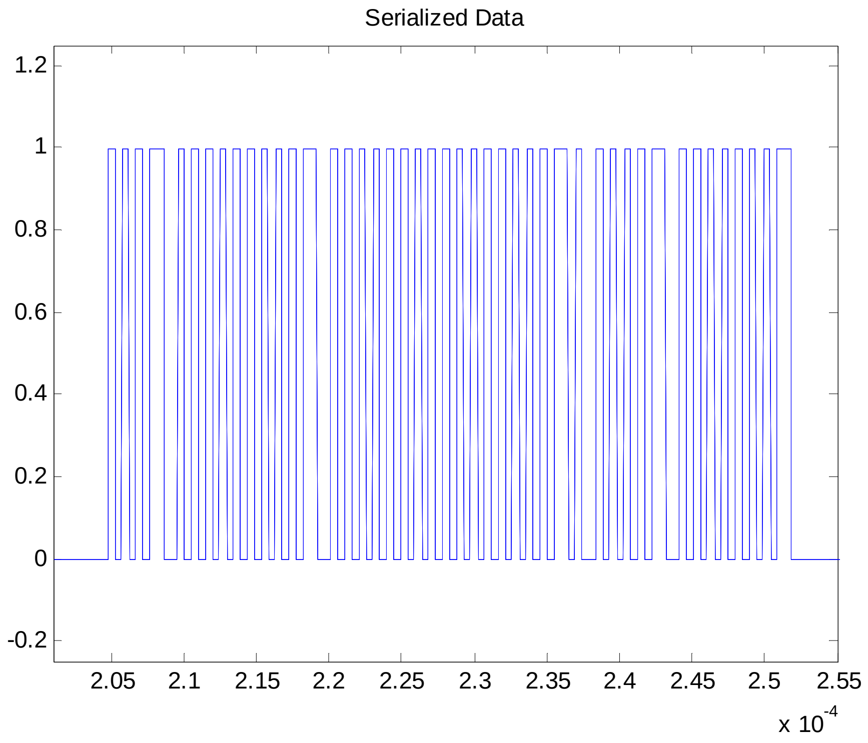


Figure 38. Serialized Data of Four Twelve Bit Constants plus a Start and a Stop Bit.

Using the Manchester decoding described in this chapter, the forty-eight bit string of data in equation (6.1) can be shown to correspond directly to the serialized string of data, minus the start and stop bit, in Figure 38.

To test the deserialization portion of the software, and to provide further confirmation that the serialization software would work in the redundant architecture design, a computer simulation was run which displayed the four initial conditions of the primary controller prior to being serialized and the four deserialized outputs in the secondary controller. The graphical representations of these four signals before and after the serialization software were then compared. The software was run for 50ms while the primary controller was moving toward steady state. This simulation represented roughly one thousand serialized strings of the four twelve bit words which provided a high degree of confidence that the software was not producing any bit errors.

The simulated results of the four initial condition values being sent into the serialization block of the primary controller and the four output values of the deserialization block in the secondary controller are shown in Figure 39. The simulated results of the graphs indicate the values sent from the primary controller were accurately interpreted by the secondary controller without any bit errors.

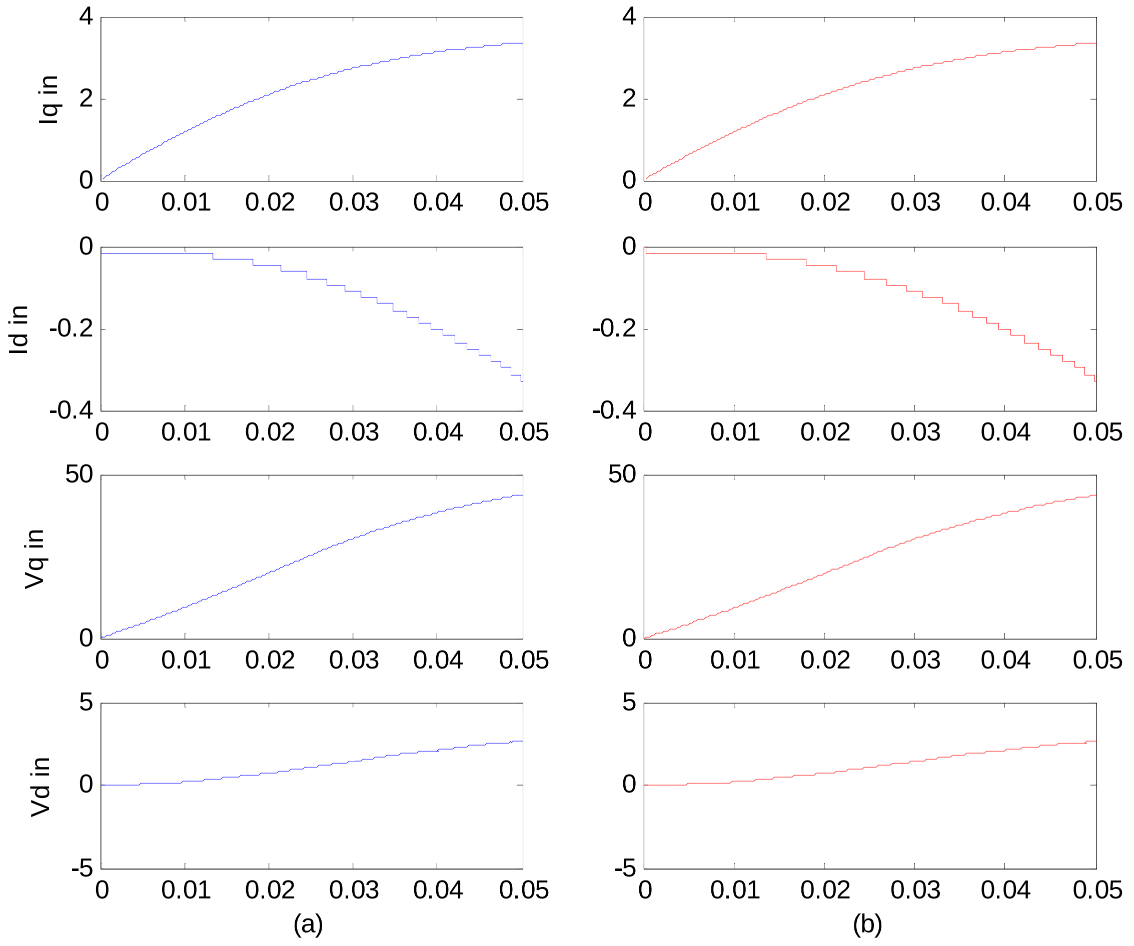


Figure 39. (a) The four integrator values of the primary controller prior to being serialized by the software. (b) The four output values of the deserialization block in the secondary controller.

2. Simulated Output Results

The simulated graph of a single phase of the output when the theta values were synchronized, and the serialized integrator values of the primary controller were passed to the secondary controller is shown in Figure 40. The same output with all three phases represented is shown in Figure 41. In these simulations the step function was set to occur at 0.05 seconds, and the graphs were zoomed in on the switching event to try and detect any minor disturbances in the sine wave. The simulated results indicated that passing the integrator values from the primary to the secondary controller through the serialization software and latching the values in the secondary controller when a fault was detected would eliminate the remaining disturbance in the software. The absence of any disturbance in the computer simulated output indicated that this was a viable solution for the redundant architecture that would meet military standards [1].

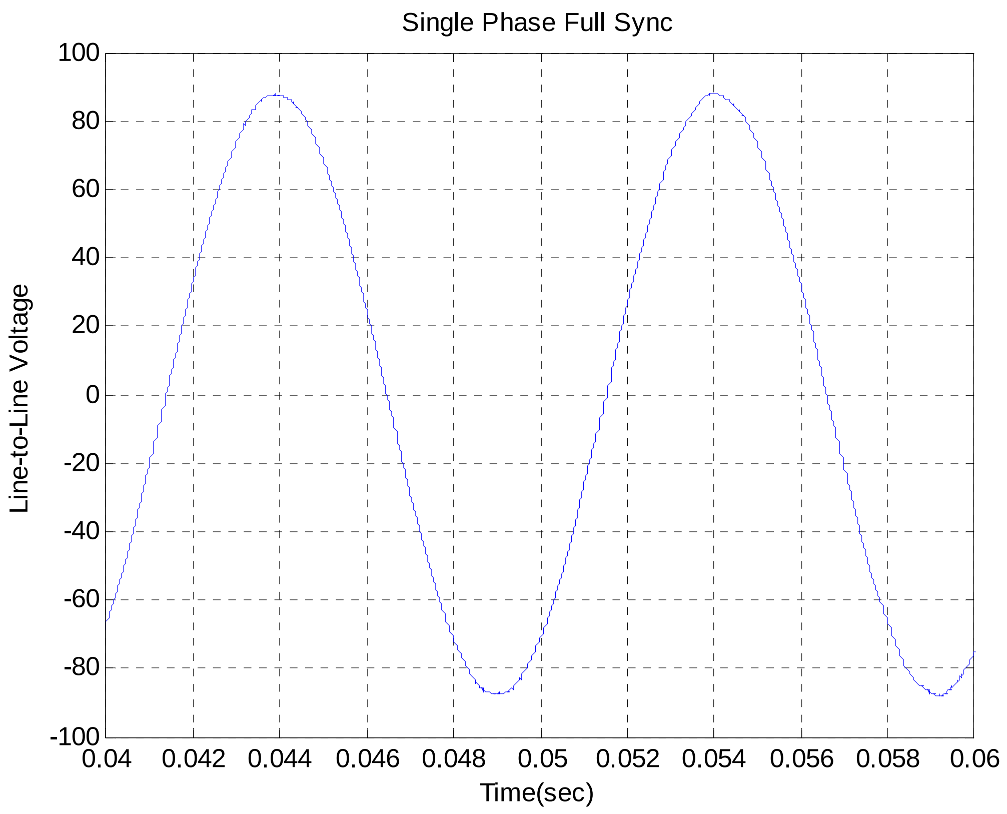


Figure 40. Simulated Single Phase Voltage Output with Theta Synchronized, Initial Conditions Passed, and the Switching Event at 0.05 seconds.

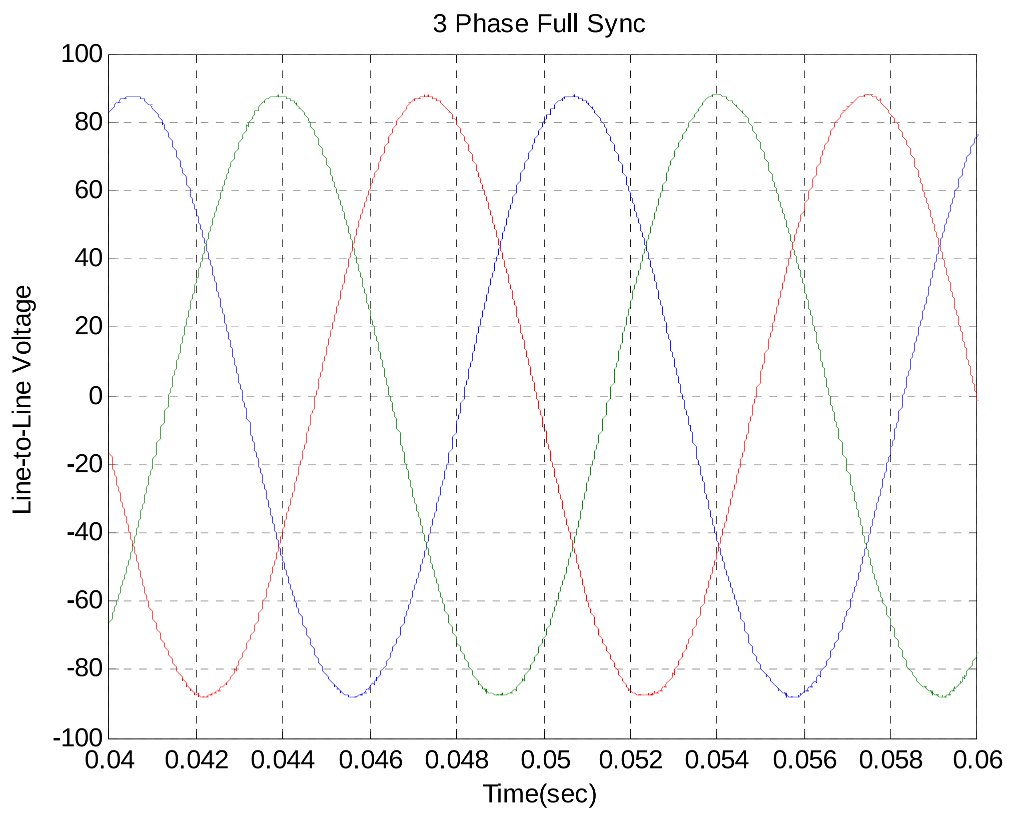


Figure 41. Simulated Three Phase Voltage Output with Theta Synchronized, Initial Conditions Passed, and the Switching Event at 0.05 seconds.

C. EXPERIMENTAL RESULTS

1. Experimental Test of Serialization Software

The first experimental test that was developed was to demonstrate that the serialized data from the primary board was actually being sent to the secondary board. To provide experimental confirmation of this, the serialization block of the software was loaded on the primary board with the same four constant values discussed in section B as its four input values and the output pin assigned to the FPGA from Table 6. The secondary board was then loaded with the deserialization block of the software along with another serialization block. This configuration on the secondary board would take the bit string from the primary board and deserialize the data back into four constant values. Those four constant values were then sent back into the serialization block on the secondary board and sent out of the board as a second string of forty-eight bits with a start and stop bit. If the data was being passed accurately from the primary to the secondary controller, and the serialization and deserialization blocks of the software were operating properly on the two FPGAs, the output string from the secondary controller should have been identical to the output string from the primary controller with roughly a 50 μs delay. The two individual bit strings from the primary and secondary boards are shown in Figure 42. These two graphs show approximately a 55 μs delay present in the output of the secondary board due to the time it took for the data string from the primary board to be deserialized into four constant values and then serialized back into a second data string.

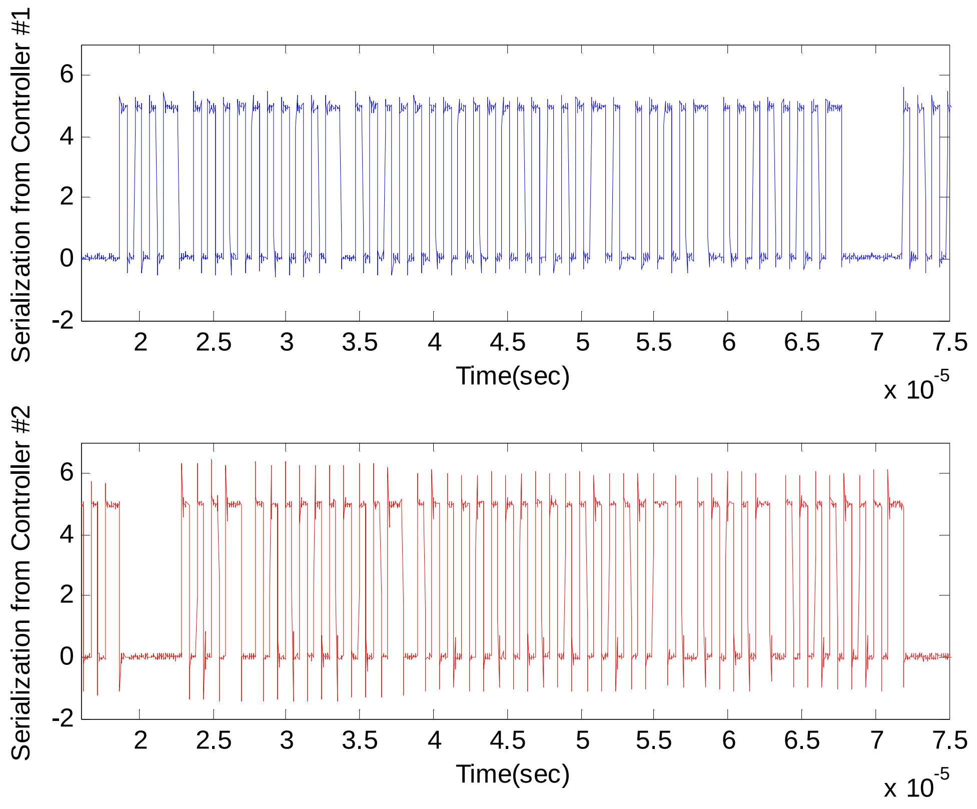


Figure 42. Serialized Data of Four Constants from the Primary and Secondary boards.

The time delay was removed and the two graphs were laid on top of each other to make it easier to see whether or not the two bit streams were the same in Figure 43. The four constant values that were serialized and sent out of the primary board were identical to the four constant values that were serialized and sent out of the secondary board. Further analysis of the graph showed that the serialized data from both of the boards corresponded to the forty-eight bit string in equation (6.1). These results experimentally confirmed the simulated serialization results as well as the ability of a data string to be passed from one physical board to another.

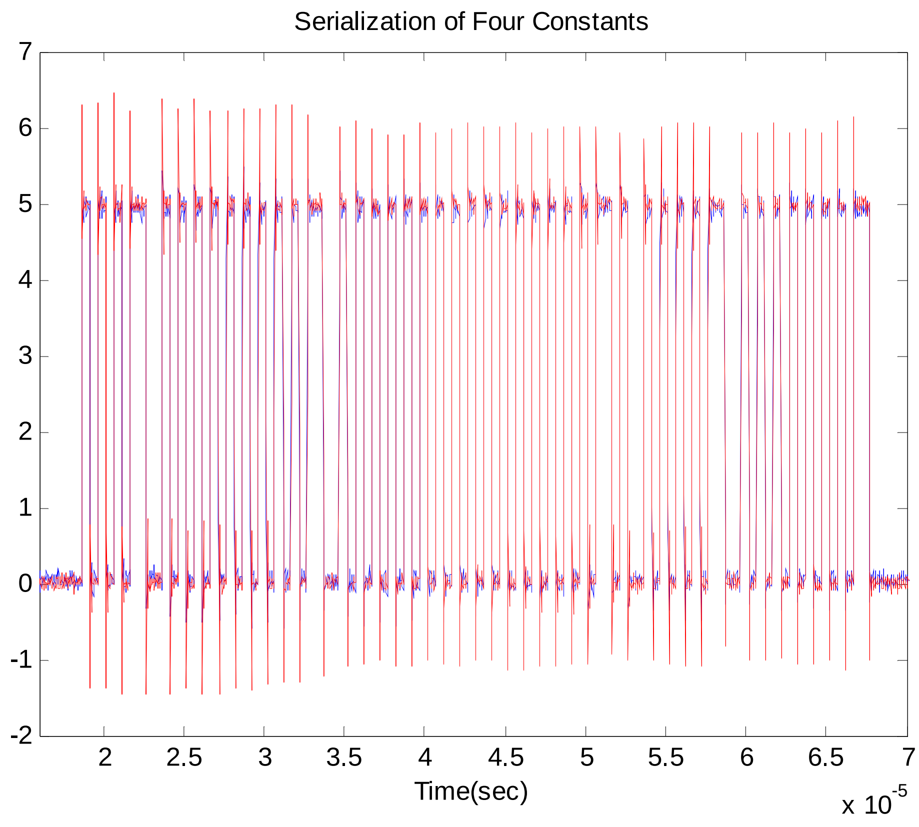


Figure 43. The Serialized Output from the Secondary Controller Superimposed on the Output from the Primary Controller

The limitations of operating Chipscope simultaneously on two separate boards prevented experimental tests from being taken similar to the simulations presented in Figure 39. It was not possible to collect data from both boards simultaneously because Chipscope could only read one board at a time through the XILINX parallel cable. The oscilloscope was also not able to be used to measure the actual integrator values because it would have required forty-eight additional test pins from each board to pass all of the information out of the FPGAs. This inability to read the internal integrator values were why the bit transfer test presented in this section was developed. This bit transfer test along with the output results of the switching event in the next section provided sufficient data that the integrator values were being transferred from the primary to the secondary board correctly.

2. Experimental Output Results

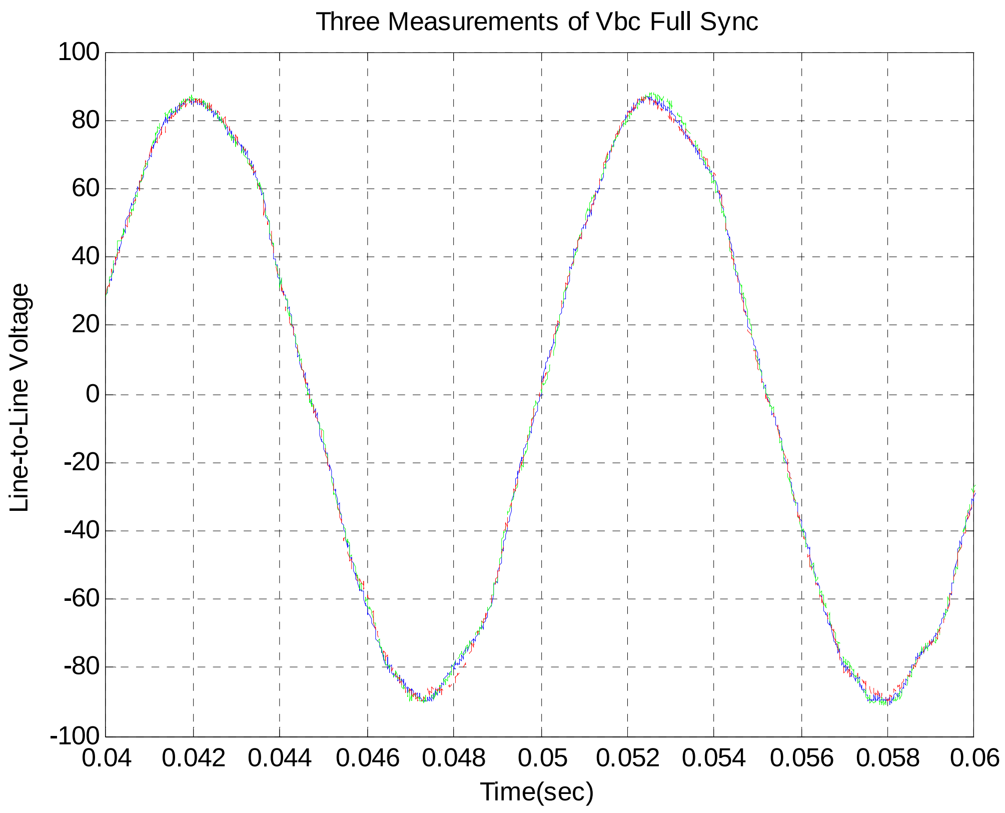


Figure 44. Three Single Phase Experimental Output Measurements with Theta Synchronized, Initial Conditions Passed, and the Switching Event at 0.05 seconds.

Three experimental measurements of a single phase of the VSI output when both the theta synchronization pulse and the initial conditions were being passed from the primary to the secondary boards are shown in Figure 44. For these experimental tests the switching control software that was used in the previous chapters was implemented in order to take multiple measurements with the switching event at the same point of the output. The switching event for the experimental results occurred at 0.05 seconds on the graph. All of the experimental measurements showed no disturbance in the VSI output during the switching event.

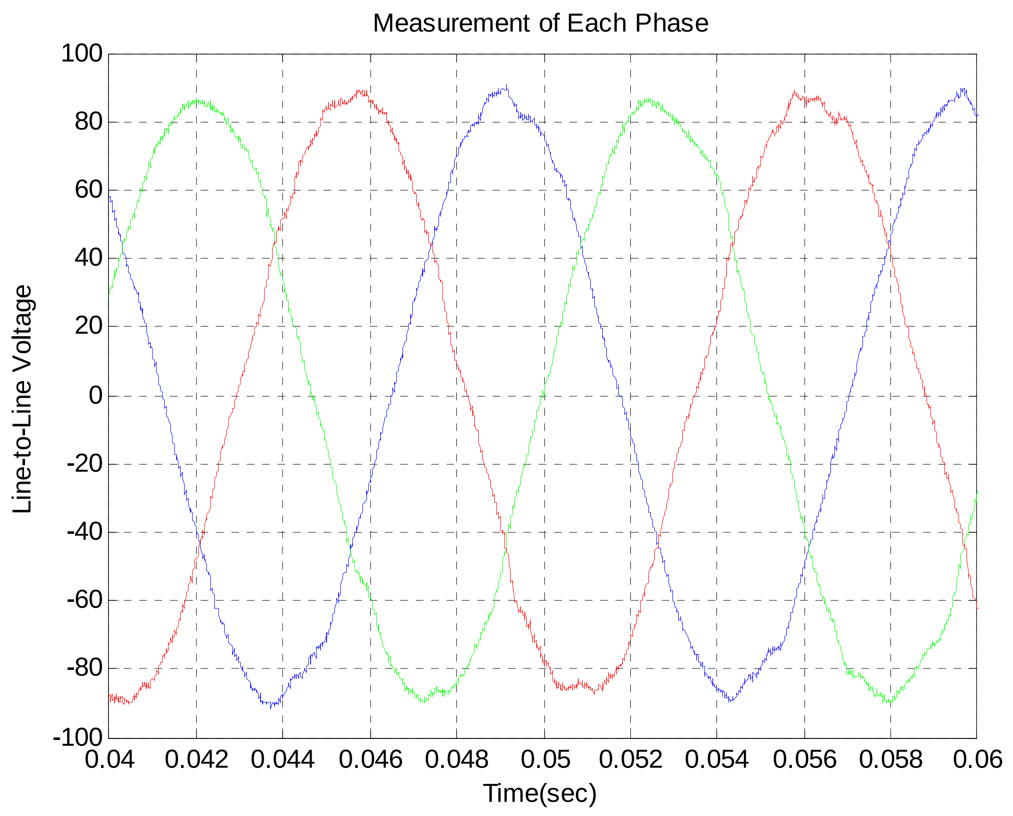


Figure 45. Experimental Measurements of the Three Phases of the VSI Output with Theta Synchronized, Initial Conditions Passed, and the Switching Event at 0.05 seconds.

The oscilloscope used for the experimental results was not able to take readings of all three phases of the output at the same time. However, since the switching event was set to trigger at the same theta point for each measurement, it was possible to take separate measurements of each phase and place them on the same graph in order to show the effects of the switching event on each phase of the output. The experimental results of all three phases of the output are shown in Figure 45. Despite the fact that the measurements had significant distortion in the voltage output, the measurements of all three phases showed no disturbance due to the switching event. Both the simulated and experimental output results provided a high level of confidence in the ability of this design to switch from a primary to a secondary controller when a fault occurred with virtually no disturbance to the output of the VSI.

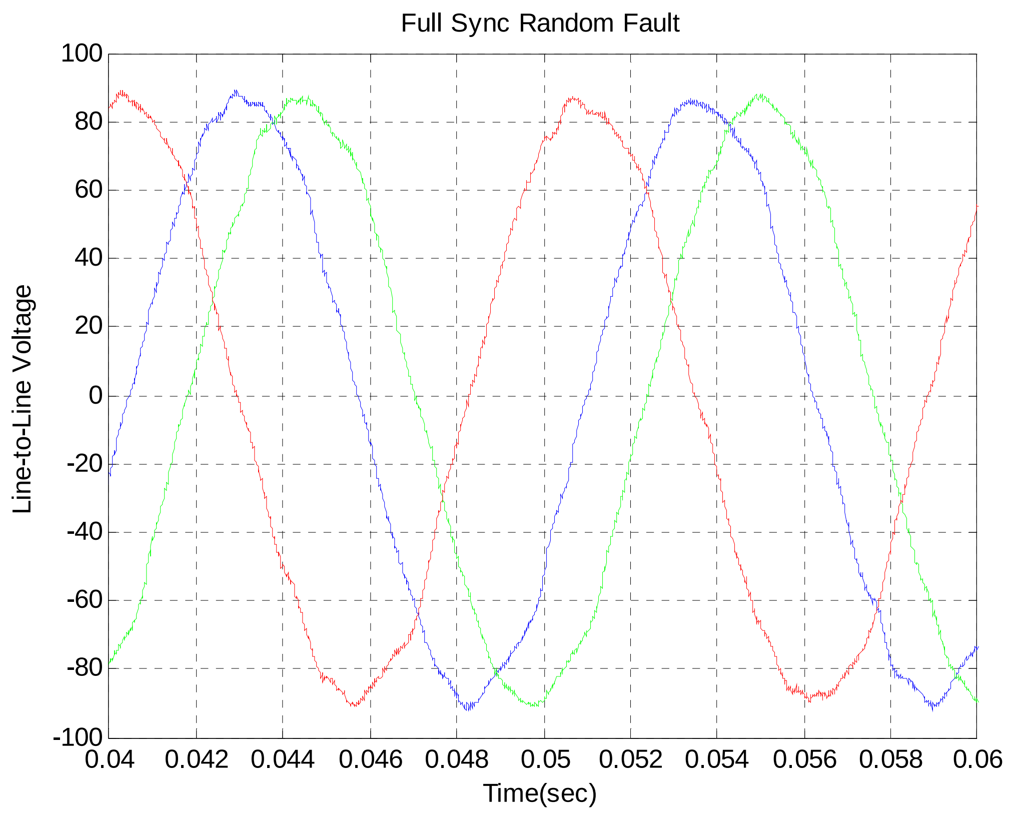


Figure 46. Three Single Phase Experimental VSI Output Measurements with Faults at random theta values and the Switching Event at 0.05 seconds.

All of the experimental results presented so far showed the switching event at a forced position of the voltage output in order to lay multiple measurements on top of each other. Since an actual redundant system would not want to force the fault signal to wait for a second condition before it switched, it was necessary to take some output measurements without the switching control software installed. Forcing the switching to occur at the same theta value every time could have also hidden possible disturbances when the system switched at an undefined theta value. In order to test this idea, three experimental measurements were taken with the switching control software block removed in order to observe the affects of the switching event on the voltage output at random points. The results of the three measurements with the switching event triggered at 0.05 seconds are shown in Figure 46. Although the distortion was still present in all of the measurements, there did not seem to be any significant disturbance in the outputs. The only minor disturbance at the 0.05 second switching event was in the red and blue graphs of the figure. However, the disturbance was so minor that it was difficult to determine if it was disturbance due to the switching or just normal distortion of the output.

D. CHAPTER SUMMARY

This chapter discussed the solution developed to be able to pass the four integrator values of the primary controller to the secondary controller using a single output pin on the primary controller and single input pin on the secondary controller. The addition of communications software that enabled the four integrator values of the primary controller to be serialized, sent through a signal bit output on the FPGA to the secondary controller, and deserialized into the four individual values to be sent to the four PI control block in the secondary controller was explained. The modifications to the PI control blocks in the secondary controller that enabled it to read the values from the primary controller and latch the last value sent when a fault was triggered was also discussed in this chapter. Simulation and experimental test results on the serialization software were presented to demonstrate its effectiveness in passing data accurately. Simulated and experimental test results of the final VSI output were also presented, which indicated that passing the integrator values from the primary to the secondary controller eliminated the remaining amplitude disturbance caused by the switching event. A degree of distortion on the output waves was also observed which could have produced some minor disturbances during certain switching events. The next chapter presents the final conclusions of this thesis along with the original contributions made by this research. Possible follow on research that could be pursed in the future is also discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis demonstrated the potential for the improved reliability of a VSI to be achieved through the implementation of operating standby redundant controller architecture. The VSI used in this thesis was designed using computer simulations that were then validated through experimental results. The closed loop space vector modulation controllers selected for this research were designed so that when a high/low fault was detected the system would automatically switch from the primary to the secondary controller. The secondary controller was designed to operate physically independent of the primary controller to reduce the risk of damage when a fault occurred, thereby providing true redundancy. After confirming that the initial redundant design worked, modifications to the software were made to allow a theta synchronization pulse from the primary to the secondary controller. This addition to the design eliminated the random phase disturbance in the VSI output during the switching event. The final step in the redundant design was the development of serialization software that would allow the four integrator values of the primary controller to be passed to the secondary controller through a single FPGA bit. Modifying the redundant software to pass the integrator values from the primary to the secondary controller to act as initial conditions when a fault occurred produced negligible disturbance in the VSI output during the switching event.

The experimental measurements of the output when both the theta pulse and the initial conditions were passed showed some distortion to the signal which might have caused some minor disturbance during some of the switching events. This distortion was most likely due to the selection of the PI gains and the hardware values. Creating a more stable system by selecting ideal hardware and gain values could help reduce the uncertainty of whether the minor disturbance during certain switching events was due to the distortion in the output signal or an issue with the data transfer from the primary to the secondary controller.

B. RECOMMENDATIONS

There are still some questions to be answered before this design could be put into any practical system. The ability of a VSI to switch to a redundant controller without any significant output disturbance provides the basis for many follow on research topics. Some possible future topics are discussed in the following sections.

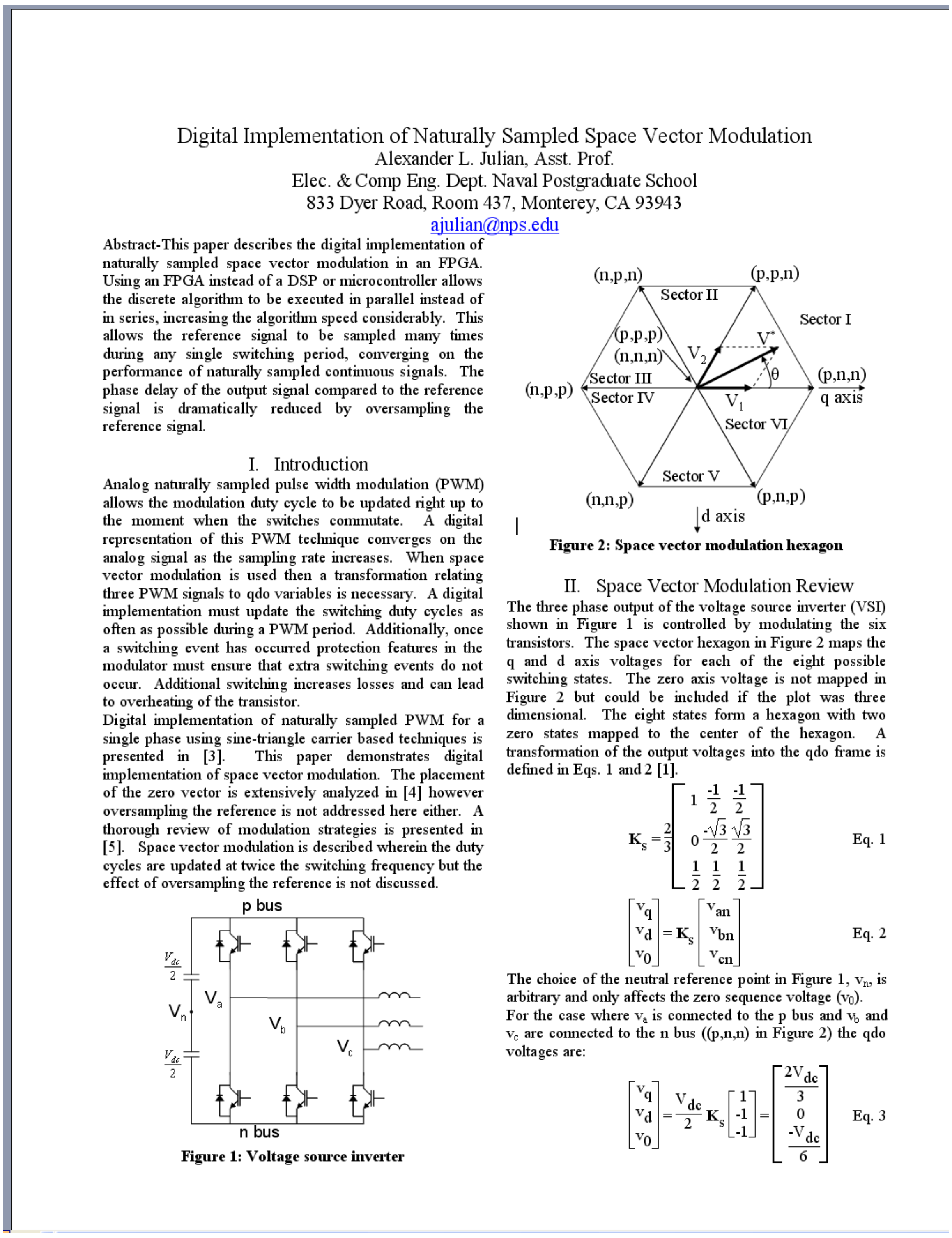
1. Reduction of the Distortion in the VSI

When the VSI output of the experimental results was viewed closely, there was significant distortion due to the gain values and noise in the system. This distortion could be reduced by determining the ideal gain and hardware values for the system. Designing a filter for the harmonic distortion could also improve the results which would help determine if there were any minor random disturbances during the switching event that were being hidden by the distortion of the output.

2. Additional Redundant Components

Demonstrating that it is possible for a VSI to switch to a redundant controller without any disturbance in its output provides the basis to implement more redundant components of the architecture presented in Chapter I of this thesis. Follow-on research could implement this redundant controller design in a VSI with a four-switch pole inverter topology.

APPENDIX A: SPACE VECTOR MODULATION NOTES



Digital Implementation of Naturally Sampled Space Vector Modulation

Alexander L. Julian, Asst. Prof.
Elec. & Comp Eng. Dept. Naval Postgraduate School
833 Dyer Road, Room 437, Monterey, CA 93943
ajulian@nps.edu

Abstract-This paper describes the digital implementation of naturally sampled space vector modulation in an FPGA. Using an FPGA instead of a DSP or microcontroller allows the discrete algorithm to be executed in parallel instead of in series, increasing the algorithm speed considerably. This allows the reference signal to be sampled many times during any single switching period, converging on the performance of naturally sampled continuous signals. The phase delay of the output signal compared to the reference signal is dramatically reduced by oversampling the reference signal.

I. Introduction

Analog naturally sampled pulse width modulation (PWM) allows the modulation duty cycle to be updated right up to the moment when the switches commutate. A digital representation of this PWM technique converges on the analog signal as the sampling rate increases. When space vector modulation is used then a transformation relating three PWM signals to qdo variables is necessary. A digital implementation must update the switching duty cycles as often as possible during a PWM period. Additionally, once a switching event has occurred protection features in the modulator must ensure that extra switching events do not occur. Additional switching increases losses and can lead to overheating of the transistor.

Digital implementation of naturally sampled PWM for a single phase using sine-triangle carrier based techniques is presented in [3]. This paper demonstrates digital implementation of space vector modulation. The placement of the zero vector is extensively analyzed in [4] however oversampling the reference is not addressed here either. A thorough review of modulation strategies is presented in [5]. Space vector modulation is described wherein the duty cycles are updated at twice the switching frequency but the effect of oversampling the reference is not discussed.

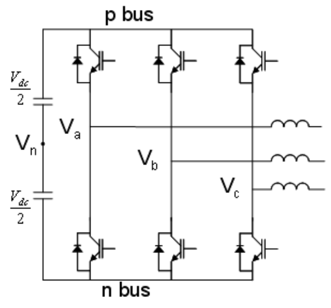


Figure 1: Voltage source inverter

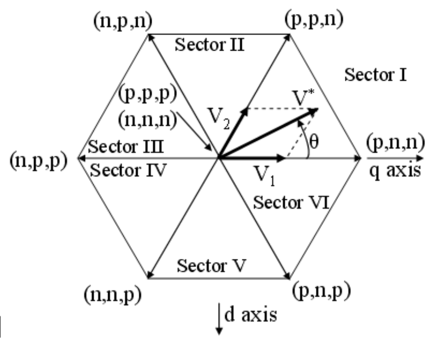


Figure 2: Space vector modulation hexagon

II. Space Vector Modulation Review

The three phase output of the voltage source inverter (VSI) shown in Figure 1 is controlled by modulating the six transistors. The space vector hexagon in Figure 2 maps the q and d axis voltages for each of the eight possible switching states. The zero axis voltage is not mapped in Figure 2 but could be included if the plot was three dimensional. The eight states form a hexagon with two zero states mapped to the center of the hexagon. A transformation of the output voltages into the qdo frame is defined in Eqs. 1 and 2 [1].

$$\mathbf{K_s} = \frac{2}{3}\begin{bmatrix} 1 & \frac{-1}{2} & \frac{-1}{2} \\ 0 & \frac{-\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \qquad \text{Eq. 1}$$

$$\begin{bmatrix} v_q \\ v_d \\ v_0 \end{bmatrix} = \mathbf{K_s}\begin{bmatrix} v_{an} \\ v_{bn} \\ v_{cn} \end{bmatrix} \qquad \text{Eq. 2}$$

The choice of the neutral reference point in Figure 1, v_n, is arbitrary and only affects the zero sequence voltage (v_0). For the case where v_a is connected to the p bus and v_b and v_c are connected to the n bus ((p,n,n) in Figure 2) the qdo voltages are:

$$\begin{bmatrix} v_q \\ v_d \\ v_0 \end{bmatrix} = \frac{V_{dc}}{2}\mathbf{K_s}\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} \frac{2V_{dc}}{3} \\ 0 \\ \frac{-V_{dc}}{6} \end{bmatrix} \qquad \text{Eq. 3}$$

Eq. 3 also defines the length of the radii forming the corners of the hexagon, 2/3 V_{dc}. For the case where v_a and v_b is connected to the p bus and v_c is connected to the n bus ((p,p,n) in Figure 2) the qdo voltages are:

$$\begin{bmatrix} v_q \\ v_d \\ v_0 \end{bmatrix} = \frac{V_{dc}}{2} \mathbf{K}_s \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} \frac{V_{dc}}{3} \\ \frac{-V_{dc}}{\sqrt{3}} \\ \frac{V_{dc}}{6} \end{bmatrix} \quad \text{Eq. 4}$$

The two states defined by Eqs. 3 and 4 form the sides of Sector I. When the reference voltage is in this sector then these two states and the zero states are used to produce an output voltage that, in the average, equals the reference voltage.

Let T_s be the total switching period, for example 100 μs when the switching frequency is 10 kHz. Let T_1 and T_2 represent the amount of time spent on states (p,n,n) and (p,p,n) respectively. The vectors V_1 and V_2 are proportional to the time spent on each state:

$$V_1 = \frac{T_1}{T_s}\frac{2\cdot V_{dc}}{3} \quad \text{Eq. 5a} \qquad V_2 = \frac{T_2}{T_s}\frac{2\cdot V_{dc}}{3} \quad \text{Eq. 5b}$$

The law of sines [2] can be used to find the duty cycles for each state:

$$\frac{2\cdot V^*}{\sqrt{3}} = \frac{V_1}{\sin(60^o\text{-}\theta)} = \frac{V_2}{\sin(\theta)} \quad \text{Eq. 6}$$

Substituting Eqs. 5a and 5b into Eq. 6 yields solutions for the time spent on each state:

$$T_1 = \frac{V^*\sqrt{3}}{V_{dc}}\cdot T_s\cdot \sin(60^o\text{-}\theta) \quad \text{Eq. 7}$$

$$T_2 = \frac{V^*\sqrt{3}}{V_{dc}}\cdot T_s\cdot \sin(\theta) \quad \text{Eq. 8}$$

Since the time spent on each state cannot exceed the total switching period then the modulation index is between zero and one:

$$mi = \frac{V^*\sqrt{3}}{V_{dc}},\ 0< mi <1,\ 0< V^* < \frac{V_{dc}}{\sqrt{3}} \quad \text{Eq. 9}$$

The amount of time spent on the zero state is the time remaining in the period:

$$T_0 = T_s\text{-}T_1\text{-}T_2 \quad \text{Eq. 10}$$

III. FPGA Implementation

After the time to spend on each state is computed then the order in which the states are applied to the load must be determined. This is the most significant difference between space vector modulation and pulse width modulation. In pulse width modulation a reference is typically compared to a triangle wave and the state transitions occur when the two signals cross. In space vector modulation there are numerous ways in which to apply the switching states to the load. When choosing a switching pattern consideration should be given to:

- Minimizing switching events
- Minimizing distortion

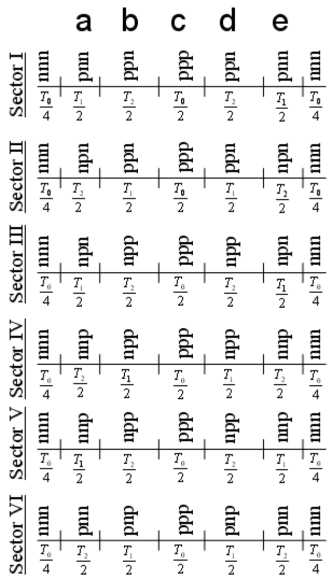

a
b
c
d
e
Sector I
nnn pnn ppn ppp ppn pnn nnn
Sector II
nnn ppn npn ppp npn ppn nnn
Sector III
nnn npn npp ppp npp npn nnn
Sector IV
nnn nnp npp ppp npp nnp nnn
Sector V
nnn nnp pnp ppp pnp nnp nnn
Sector VI
nnn pnn pnp ppp pnp pnn nnn

Figure 3: Switching Pattern for each sector (switching state on top, time duration on the bottom for each sector description)

The FPGA implementation presented here accomplishes the pattern shown in Figure 3. Of particular note is that the order of the states used in each sector changes so that there is only one transistor switching at each state transition. The state associated with timer T_1 is the first non-zero state in Sectors I, III & V. The state associated with timer T_2 is the first non-zero state in Sectors II, IV & VI. This factor is accounted for in the gate signal generation logic (Figure 5) by switching the timers used to identify the point in the modulation period. The distinct times in the modulation period are identified with the letters a to e in Figure 3. The synthesis of the logic that determines the time within the modulation period is shown in Figure 5.

The input to the modulator is the reference voltage in Cartesian coordinates. The Cartesian coordinates are used since outer control loops typically operate on the q and d axis voltages to generate a reference for control. The Cartesian to polar transformation is accomplished using a cordic algorithm, provided by XILINX inside the System Generator toolbox.

For the implementation demonstrated in this paper the ramp counter (Figure 4 & Figure 5) increments every clock cycle. The oscillator runs at 24 MHz and the switching frequency is 10 kHz so the ramp counter resets after counting up to 2400. In Figure 4 the sector of the reference voltage is identified by comparing the angle, θ, to the boundaries of the hexagon which are 0°, 60°, 120°, 180°, 240° & 300°.

IV. Digital Natural Sampling Modulation

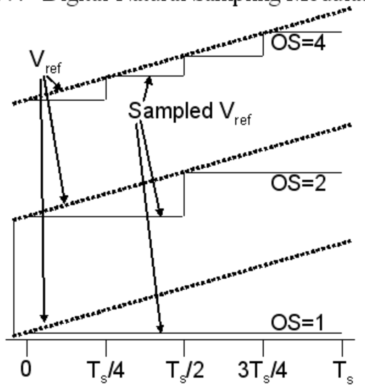


Fig. M: Oversampled reference signal when OS=1, 2 and 4.

Fig. M shows the sampled reference signal that is used to compute the PWM timer values in the fgpa. The duty cycles, T_0, T_1 and T_2, are updated at the same instant that the reference signals are resampled.

The sample and hold circuit of Figure 4 allows the digital modulation to behave like a naturally sampled system. Typical modulation updates the sampled signal when the ramp counter resets to zero. Double update modulation would update the sampled signals when the ramp counter is at zero and ½ of the final count value. As the update rate of the sampled signals increases the discrete system converges on the behaviour of a continuous system provided that the entire discrete algorithm is being updated at a very high rate. Using an FPGA instead of a DSP or microcontroller allows the discrete algorithm to be executed in parallel instead of in series, increasing the algorithm speed considerably. In the example presented here, the timers are recomputed 16 times each PWM period. A typical DSP could not achieve this algorithm update rate.

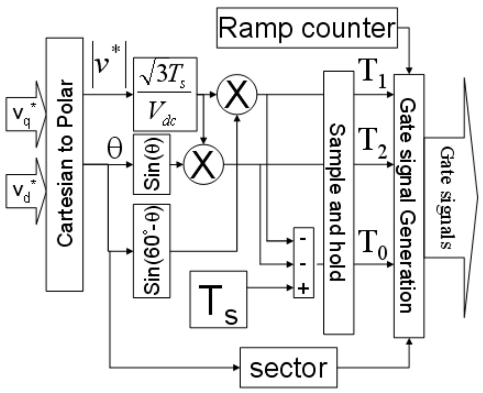


Figure 4: Space Vector Modulation Algorithm

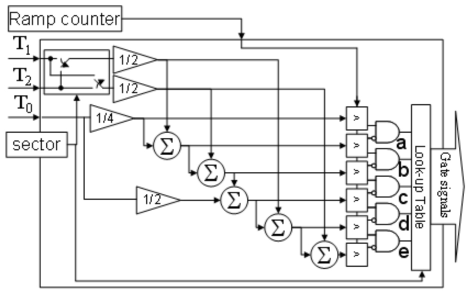


Figure 5: Gate Signal Generation Block (see Figure 4)

When the sample and hold circuit in Figure 4 updates the timer values much faster than the ramp counter period then it is possible for the gate signals to switch more than once each switching period if the reference is changing very quickly. This can lead to transistor overheating and failure. Rather than excessively bandlimit the reference signals a lockout state machine is used to protect the transistors. The gate signals are sent to a state machine, shown in Figure 6, that allows only one switching event every period. When the ramp counter equals zero then the output of the state machine equals the input, which is the gate signal from Figure 5. This occurs during the zero state, *nnn*. When the input gate signal, *x*, transitions from high to low then the state machine output, *y*, remains fixed until the end of the ramp counter period. Referring to Figure 3, this event occurs at the end of modulation period c, d or e and the gate signal remains unchanged for the rest of the period.

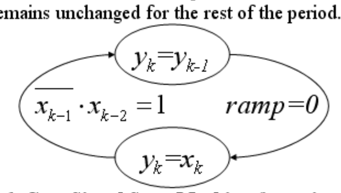


Figure 6: Gate Signal State Machine (input is *x*, output is *y*)

V. Algorithm size in FPGA

Together Figure 4, Figure 5 and Figure 6 describe the discrete algorithm embedded in an FPGA. The target FPGA used for this research is the Virtex-II XC2V1000. This chip has 1 million gates. The space vector modulation algorithm and a sine wave for v_q^* and v_d^* uses an equivalent gate count of 250,000 gates, according to the ISE Foundation mapping report, when designed using System Generator and complied using ISE Foundation. The reference voltages are 12 bits. The lookup sine table is 2^{10} 12 bit words. The math was accomplished with at least 12 bits of numerical precision throughout.

VI. Experimental Results

These experimental results measure the voltage gain and phase versus frequency as a function of the modulation index when the sample and hold rate of the timers is varied. The PWM frequency used in these experiments is 10 kHz.

The gain that is measured is from the output gate signals to the reference, v_α^* (Fig. 4). The reference is sent out through a D/A converter operating at 93.75 kHz. The analog signal is then measured with a differential voltage probe that has an RC input filter. The gate signals, V_{ab}, are measured by a differential voltage probe with an identical RC filter on its input. This creates 2 sinusoidal signals that can be compared. Finally, the RMS values of the signals and the phase between them is measured using functions in the 'measure menu' of a Tektronix TDS3014B oscilloscope.

Figure 7 shows that the gain is not greatly affected by the oversampling of the reference signals however some additional gain is realized by oversampling (the gain is 0.88 instead of 0.83). The impact of oversampling is more significant in the phase of the output voltage compared to the reference, as shown in Figure 8. For a sinusoidal reference at 2 kHz the phase of the output lags by almost 45° at 2 kHz when oversampling is not used. When the reference signal is oversampled 16 times each PWM period then this phase lag is reduced to 7°. Figures 9 and 10 show the measured voltage gain and phase when the modulation index is 0.75 instead of one.

The curve when OS=2, which is widely used (double update PWM), falls between OS=1 and OS=4 in Figs. 7-10. The final paper will include a comparison of the spectra for different oversampling rates. This information could not be included due the digest size limitation.

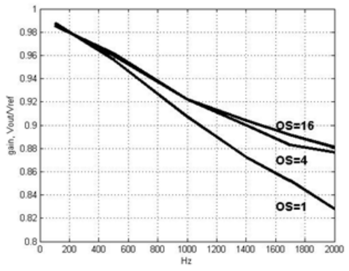


Figure 7: Voltage gain for different oversampling rates (OS) when the modulation index (*mi*) is one.

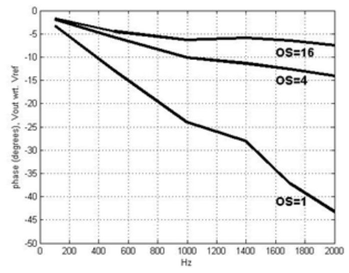


Figure 8: Phase difference for different oversampling rates (OS) when *mi* is one.

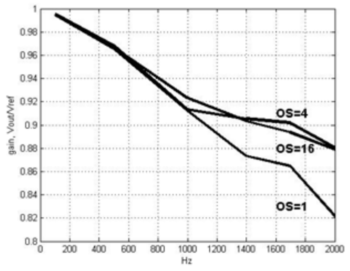


Figure 9: Voltage gain for different oversampling rates (OS) when *mi* is 0.75.

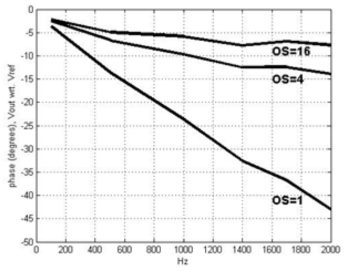


Figure 10: Phase difference for different oversampling rates (OS) when the modulation index (*mi*) is 0.75.

VII. Conclusion

Embedding space vector modulation in an FPGA creates an opportunity to oversample the reference signal. This digital implementation of naturally sampled space vector modulation significantly improves the voltage gain and phase of the modulator compared to the reference signal. This improvement can enhance the performance of wide bandwidth voltage source inverters.

VIII. Acknowledgments

The ***Xilinx University Program*** generously provided ISE Foundation and System Generator software which were used to embed the algorithms in an FPGA.

IX. References

[1] "Analysis of Electric Machinery and Drive Systems", Paul C. Krause, et al, IEEE Press, 2002.

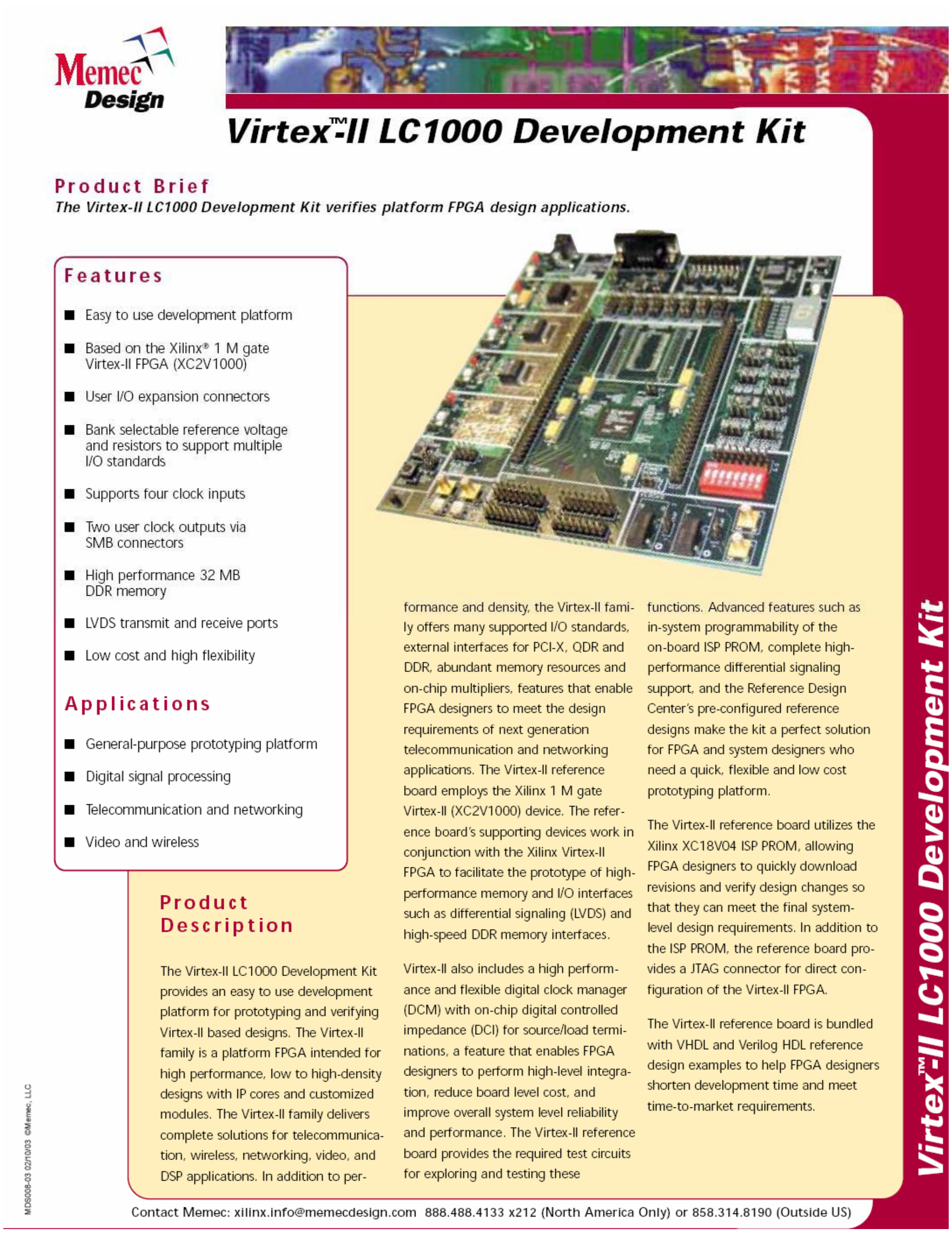
[2] "Space Vector Modulated Three-Phase to Three-Phase Matrix Converter with Input Power Factor Correction", Laszlo Huber et al, IEEE Trans. On Ind. App., Vol. 31, No. 6, Nov/Dec 1995.

[3] "Digitally-Implemented Naturally Sampled PWM Suitable for Multilevel Control", Geoffrey R. Walker, IEEE Trans. on Power Electronics, Vol. 18, No. 6, Nov 2003.

[4] "The Significance of Zero Space Vector Placement for Carrier-Based PWM Schemes", Donald Grahame Holmes, IEEE Trans. On Ind. App., Vol. 32, No. 5, Sep/Oct 1996.

[5] "Pulsewidth Modulation-A Survey", Joachim Holtz, IEEE TRANS ON IND ELEC, VOL. 39, NO. 5, DEC 1992.

APPENDIX B: VIRTEX II FPGA

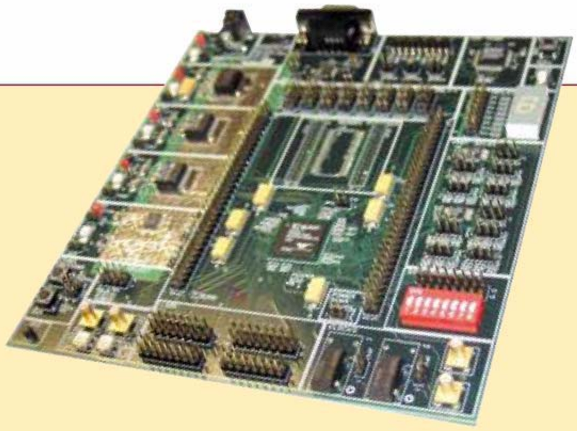


Memec Design

Virtex™-II LC1000 Development Kit

Product Brief

The Virtex-II LC1000 Development Kit verifies platform FPGA design applications.



Features

- Easy to use development platform
- Based on the Xilinx® 1 M gate Virtex-II FPGA (XC2V1000)
- User I/O expansion connectors
- Bank selectable reference voltage and resistors to support multiple I/O standards
- Supports four clock inputs
- Two user clock outputs via SMB connectors
- High performance 32 MB DDR memory
- LVDS transmit and receive ports
- Low cost and high flexibility

Applications

- General-purpose prototyping platform
- Digital signal processing
- Telecommunication and networking
- Video and wireless

Product Description

The Virtex-II LC1000 Development Kit provides an easy to use development platform for prototyping and verifying Virtex-II based designs. The Virtex-II family is a platform FPGA intended for high performance, low to high-density designs with IP cores and customized modules. The Virtex-II family delivers complete solutions for telecommunication, wireless, networking, video, and DSP applications. In addition to performance and density, the Virtex-II family offers many supported I/O standards, external interfaces for PCI-X, QDR and DDR, abundant memory resources and on-chip multipliers, features that enable FPGA designers to meet the design requirements of next generation telecommunication and networking applications. The Virtex-II reference board employs the Xilinx 1 M gate Virtex-II (XC2V1000) device. The reference board's supporting devices work in conjunction with the Xilinx Virtex-II FPGA to facilitate the prototype of high-performance memory and I/O interfaces such as differential signaling (LVDS) and high-speed DDR memory interfaces.

Virtex-II also includes a high performance and flexible digital clock manager (DCM) with on-chip digital controlled impedance (DCI) for source/load terminations, a feature that enables FPGA designers to perform high-level integration, reduce board level cost, and improve overall system level reliability and performance. The Virtex-II reference board provides the required test circuits for exploring and testing these functions. Advanced features such as in-system programmability of the on-board ISP PROM, complete high-performance differential signaling support, and the Reference Design Center's pre-configured reference designs make the kit a perfect solution for FPGA and system designers who need a quick, flexible and low cost prototyping platform.

The Virtex-II reference board utilizes the Xilinx XC18V04 ISP PROM, allowing FPGA designers to quickly download revisions and verify design changes so that they can meet the final system-level design requirements. In addition to the ISP PROM, the reference board provides a JTAG connector for direct configuration of the Virtex-II FPGA.

The Virtex-II reference board is bundled with VHDL and Verilog HDL reference design examples to help FPGA designers shorten development time and meet time-to-market requirements.

Virtex™-II LC1000 Development Kit

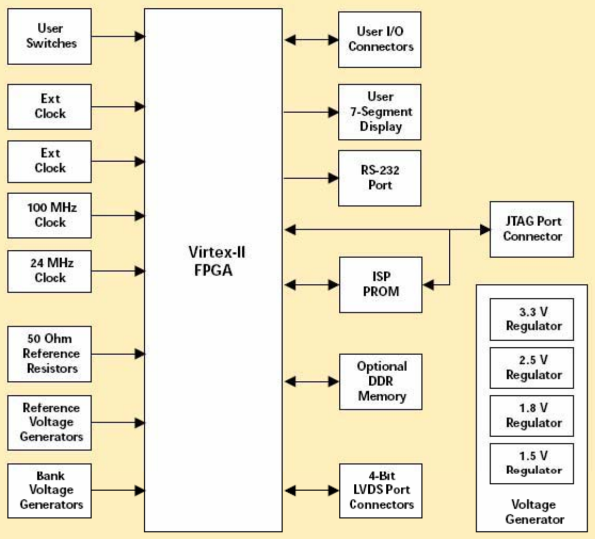
MDS008-03 02/10/03 ©Memec, LLC

Contact Memec: xilinx.info@memecdesign.com 888.488.4133 x212 (North America Only) or 858.314.8190 (Outside US)

Memec Design

Virtex-II LC1000 Development Kit Includes:

- Virtex-II Reference Board
  - 1 M gate Virtex-II FPGA device (XC2V1000-4FG256C)
  - 3.3 V, 2.5 V, 1.8 V and 1.5 V on-board voltage regulators
  - XC18V04 ISP PROM
  - On-board 32 MB DDR memory
  - RS-232 port
  - User switches and user I/O ports
  - Seven-segment LED display
  - LVDS transmit and receive ports
  - Eight-bank SelectI/O voltage (VCCO) settings (1.5 V, 1.8 V, 2.5 V and 3.3 V options)
  - Eight-bank reference voltage (VREF) settings (1.5 V, 1.25 V, 1.0 V, 0.90 V and 0.75 V options)
  - Bank reference resistors to support DCI
  - Two on-board clock sources and two SMB user clock inputs
  - Two user SMB clock output connectors
  - JTAG port
- AC-to-DC power supply adapter
- Complete reference designs with source code (VHDL and Verilog HDL)
- Bundled software options



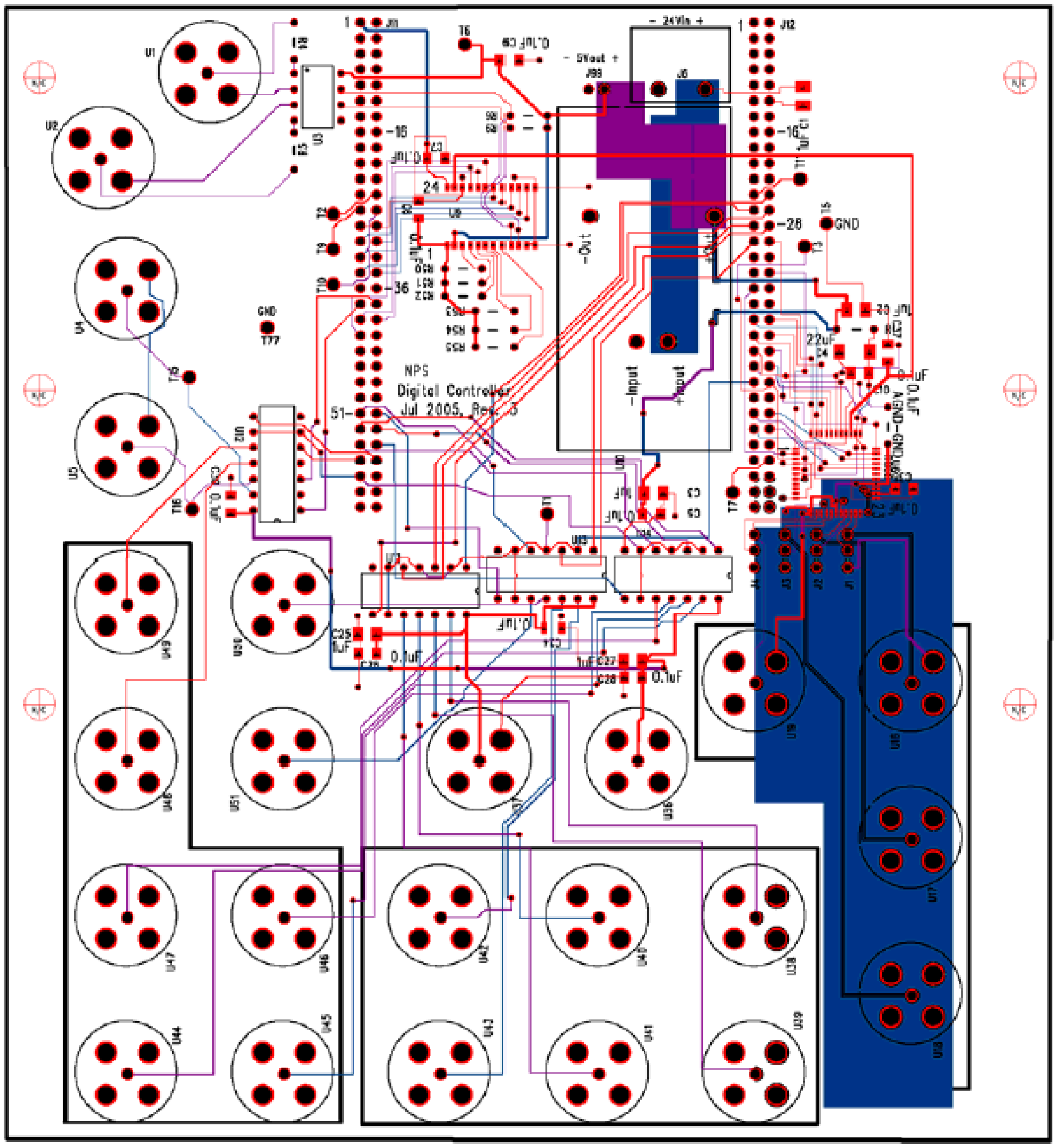
Ordering Information

	Americas Part #	International Part #
Virtex-II LC1000 Development Kit		
Virtex-II Development Kit	DS-KIT-V2LC1000	DS-KIT-V2LC1000-EURO
Kit with ISE Alliance and JTAG Cable	DS-KIT-V2LC1000-ALI	
Kit with ISE Foundation and JTAG Cable	DS-KIT-V2LC1000-ISE	

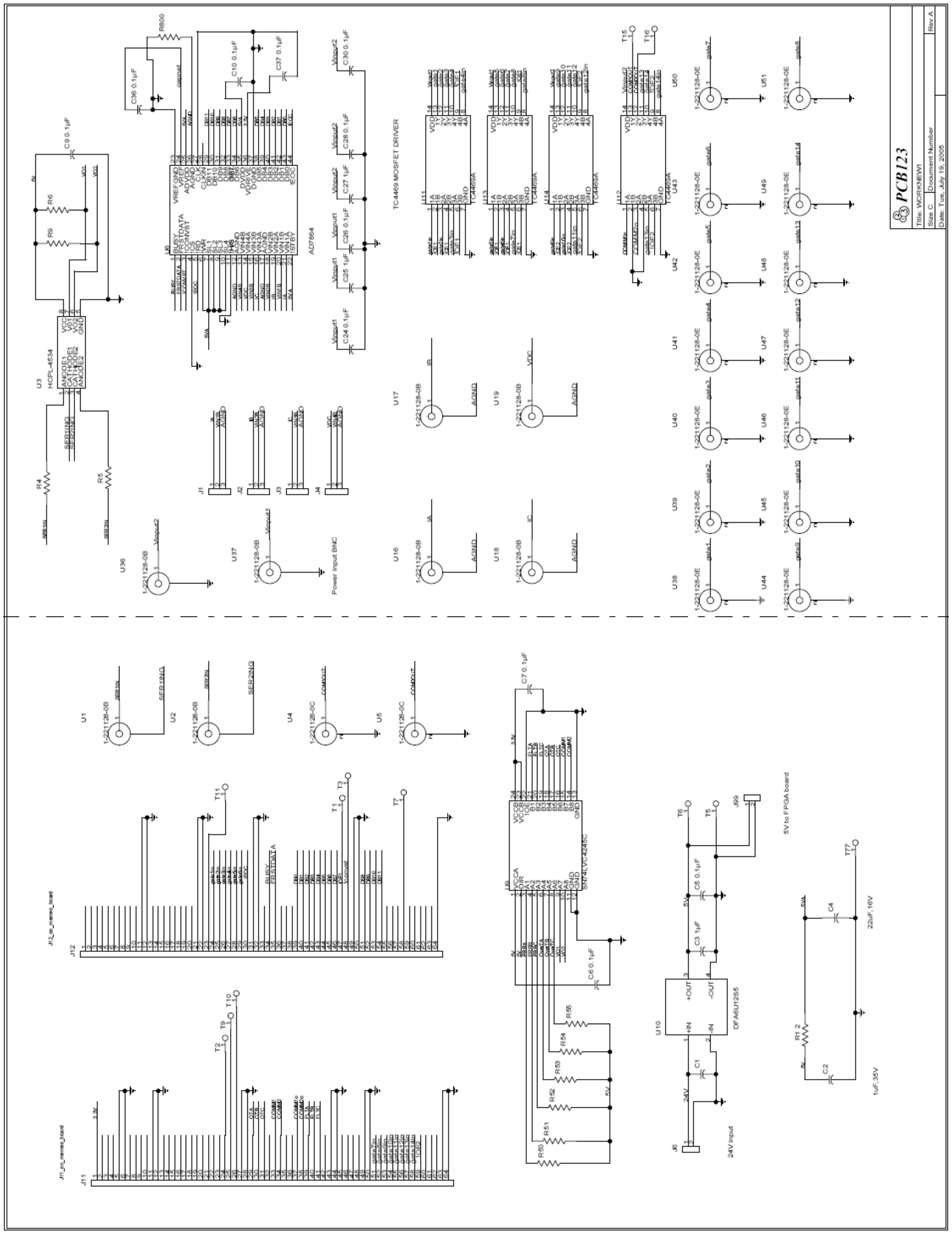
Virtex™-II LC1000 Development Kit

APPENDIX C: CUSTOMIZED INTERFACE BOARD

A. PCB 123 LAYOUT

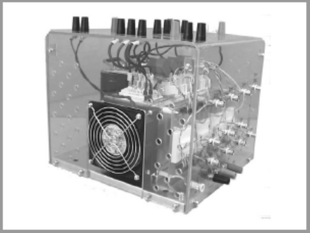


B. SCHEMATIC



APPENDIX D: THREE PHASE RECTIFIER

SEMISTACK - IGBT



SEMITRANS Stack[1)]

Three-phase rectifier + inverter with brake chopper

SEMITEACH - IGBT
SKM 50 GB 123D
SKD 51
P3/250F

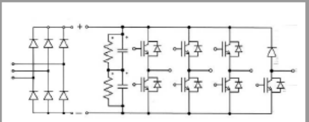
Features

- Multi-function IGBT converter
- Transparent enclosure to allow visualization of every part
- IP2x protection to minimize safety hazards
- External banana/BNC type connectors for all devices
- Integrated drive unit offering short-circuit detection/cut-off, power supply failure detection, interlock of IGBTs + galvanic isolation of the user
- Forced-air cooled heatsink

Typical Applications

- Education: One stack can simulate almost all existing industrial applications:
- 3-phase inverter+brake chopper
- Buck or boost converter
- Single phase inverter
- Single or 3-phase rectifier

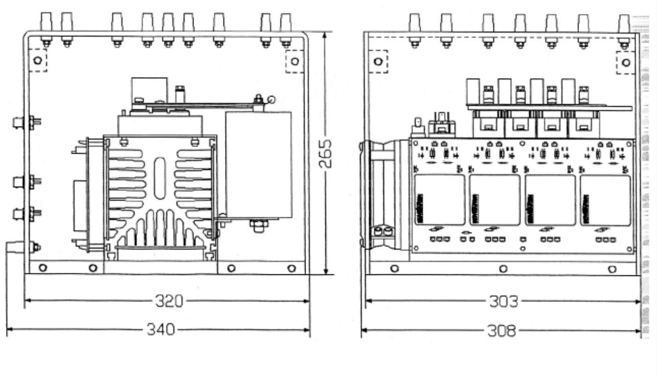
[1)] Photo non-contractual



B6U + B6CI + E1CIKF

Circuit	I_{rms} (A)	V_{ac} / V_{dcmax}	Types
B6CI	30	440 / 750	SEMITEACH - IGBT

Symbol	Conditions	Values	Units
I_{rms}	no overload	30	A
	IGBT - 4x SKM 50 GB 123D		
V_{CES}		1200	V
$V_{CE(SAT)}$	I_c= 50A, V_{GE}= 15V, chip level; T_j= 25(125)°C	2,7 (3,5)	V
V_{GES}		±20	V
I_C	T_{case}= 25 (80)°C	50 (40)	A
I_{CM}	T_{case}= 25 (80)°C; t_p= 1ms	100 (80)	A
	Rectifier - 1x SKD 51/14		
$V_{in(max)}$	without filter	3 x 480	V
	with filter	3 x 380	V
	DC Capacitor bank - Electrolytic 2x 2200µF/400V		
C_{eqvl}	total equivalent capacitance	1100 / 800	µF / V
V_{DCmax}	max. DC voltage applied to the capacitor bank	750	V
	Driver - 4x SKHI 22		
Power supply		0 / 15	V
Current consumption	max; per driver	16	mA
Thermal trip	Normally Open type (NO)	71	°C

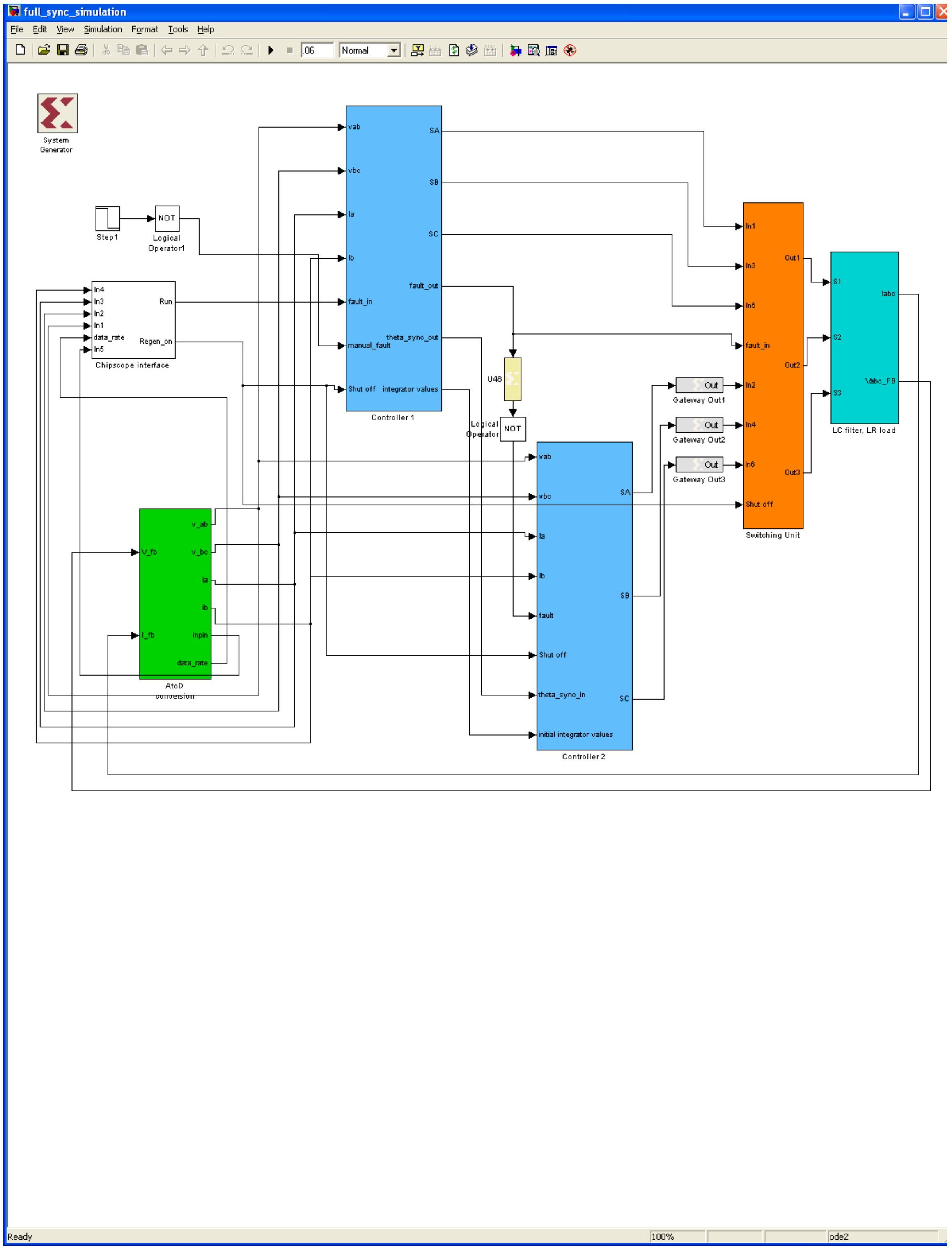


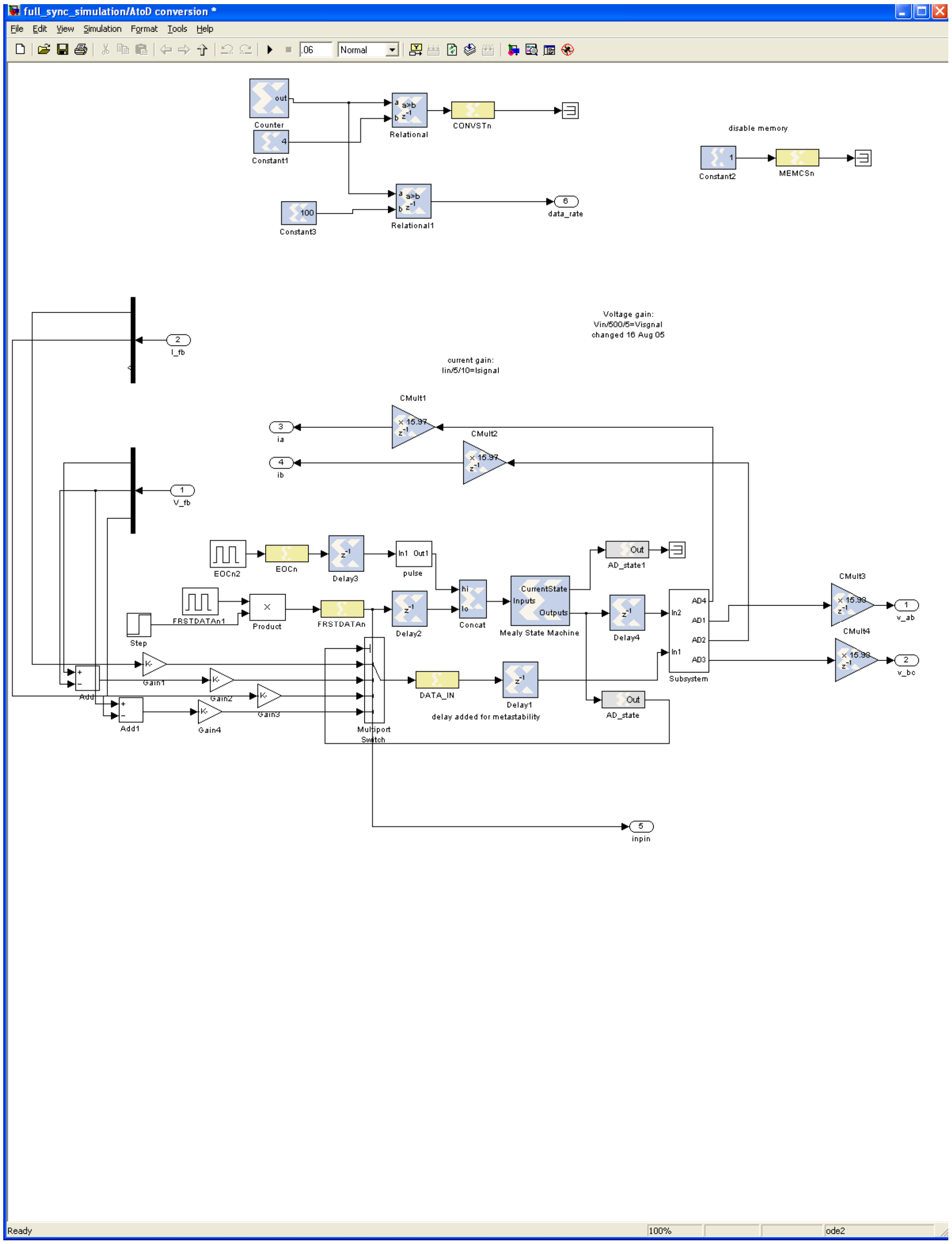
General dimensions

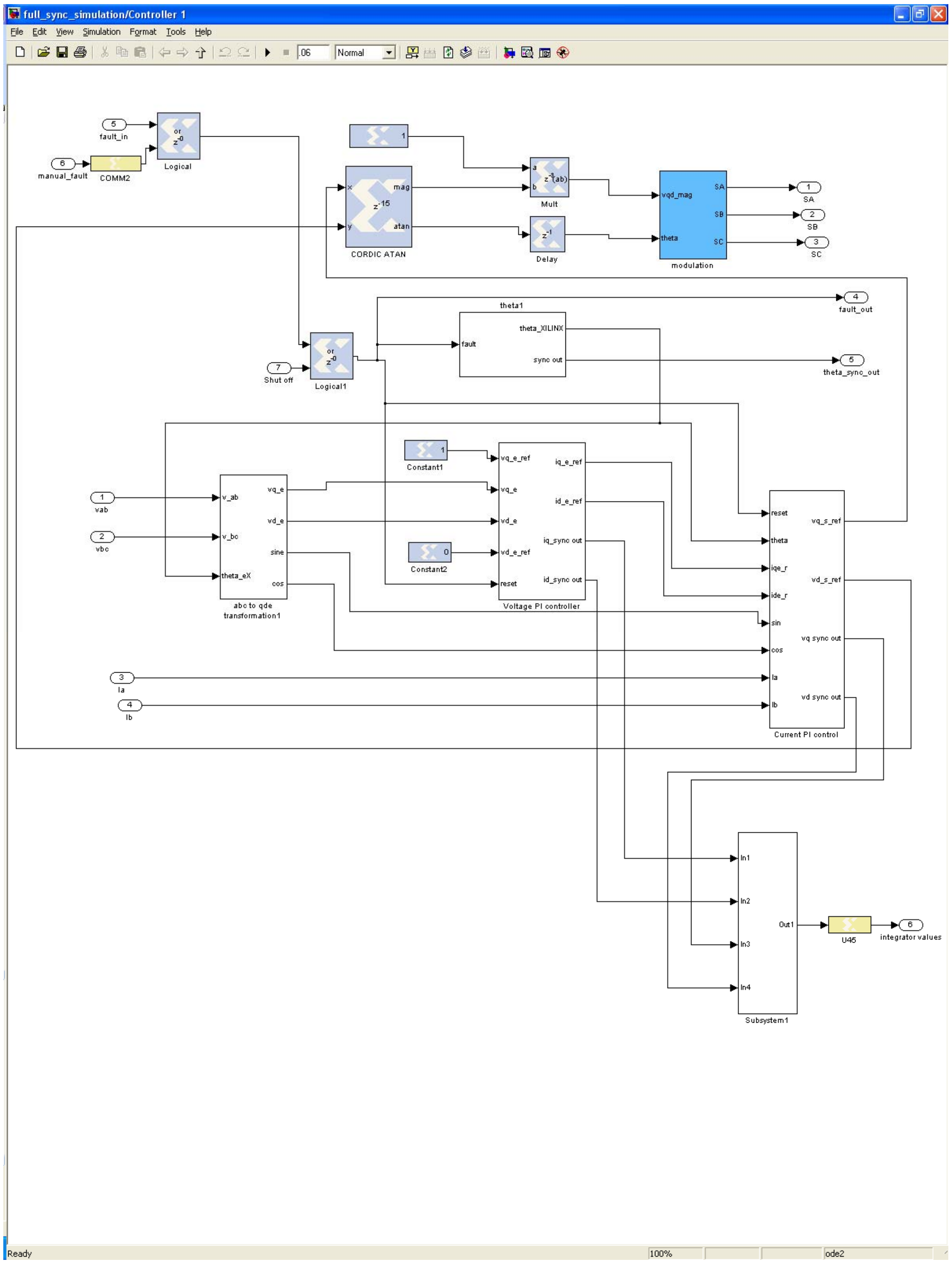
This technical information specifies semiconductor devices but promises no characteristics. No warranty or guarantee expressed or implied is made regarding delivery, performance or suitability.

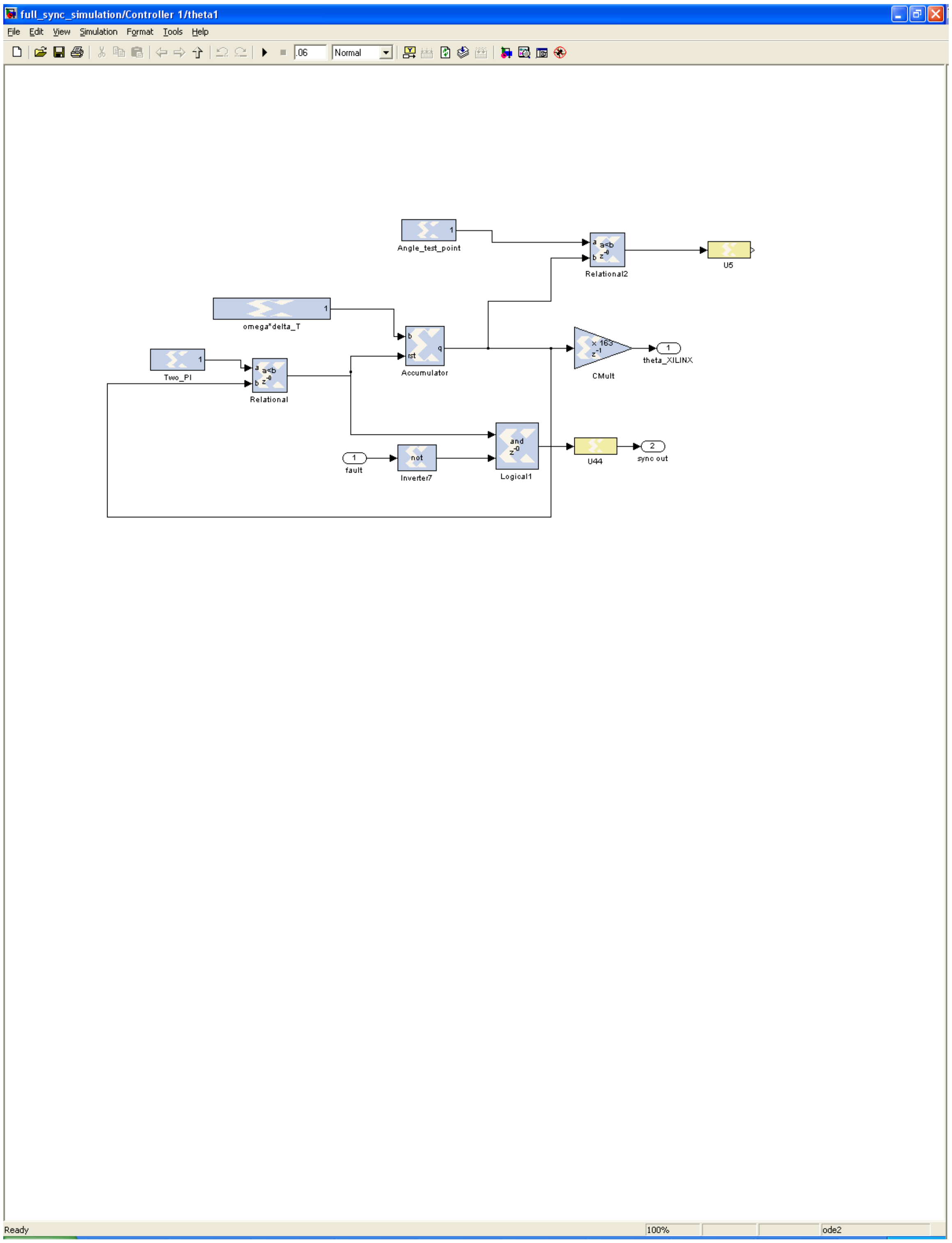
THIS PAGE INTENTIONALLY LEFT BLANK

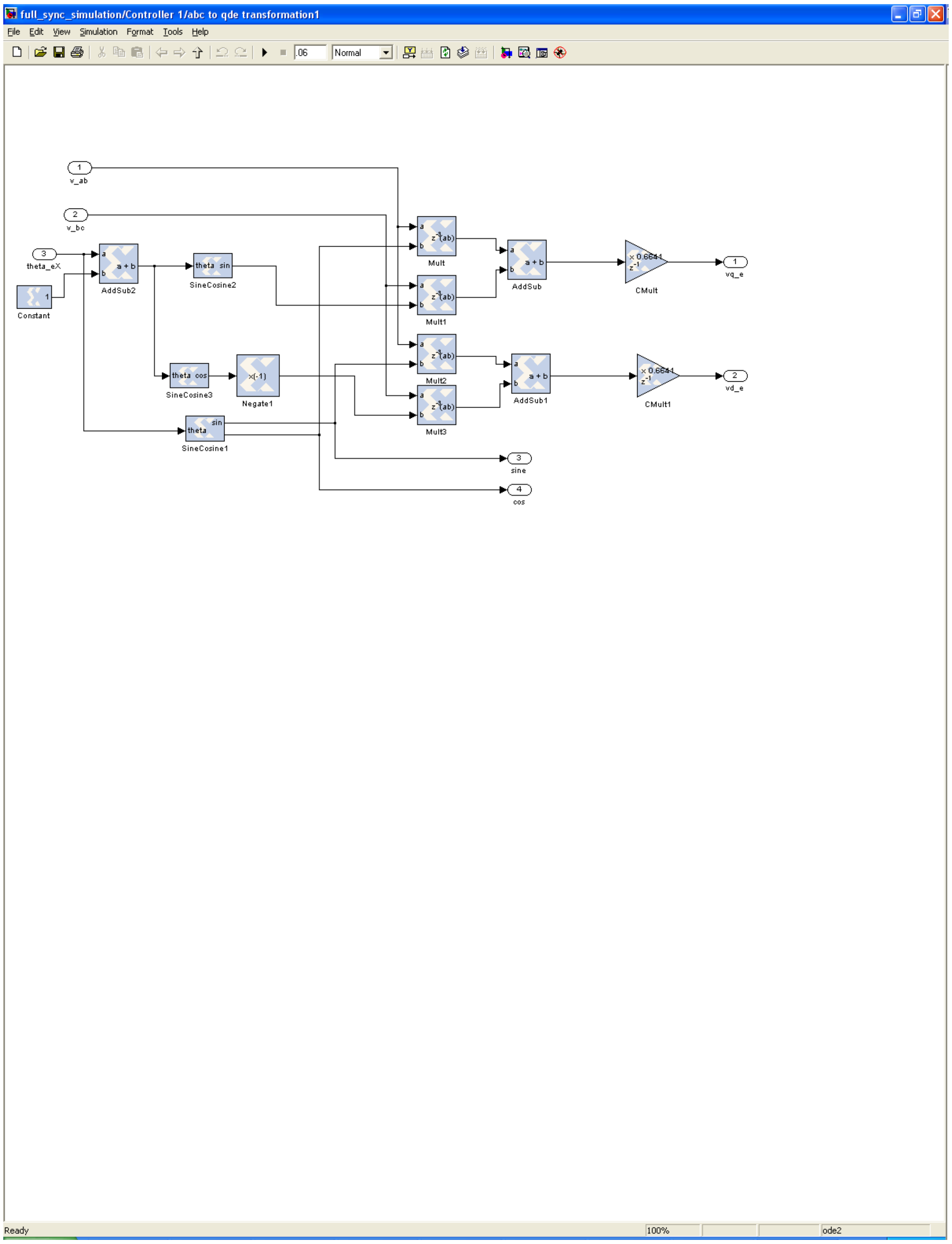
APPENDIX E: XILINX BLOCK CODE

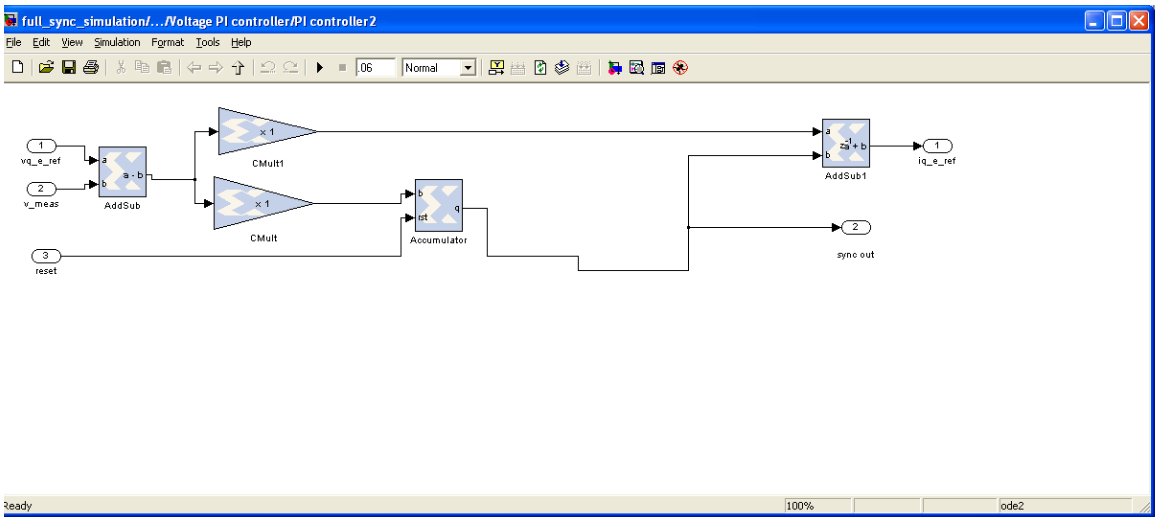


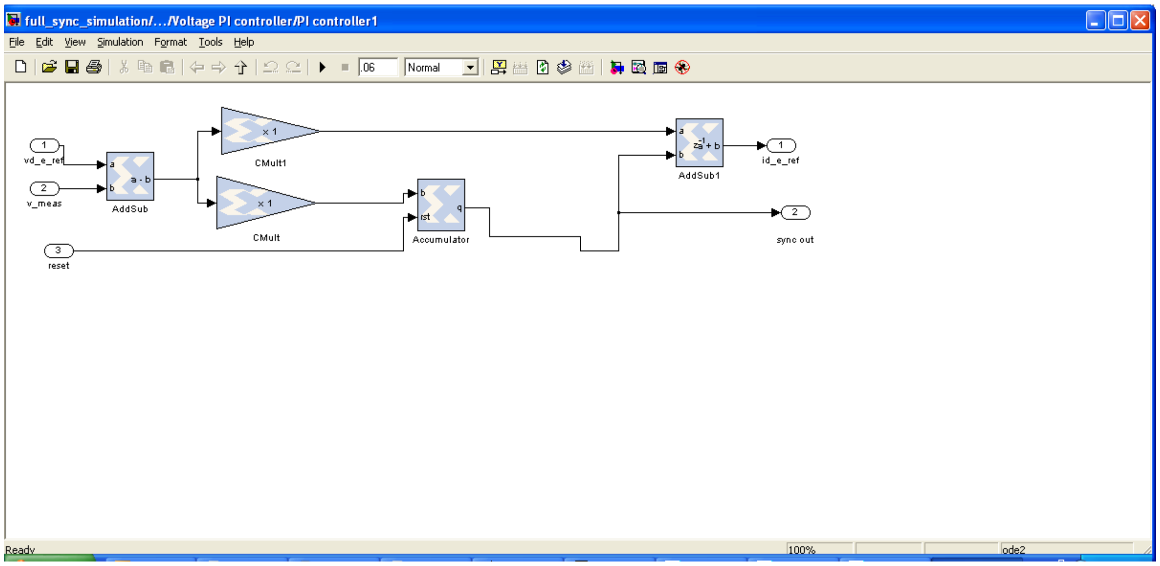

full_sync_simulation/AtoD conversion *
File Edit View Simulation Format Tools Help
.06
Normal
Counter
Constant1
Relational
CONVSTn
disable memory
Constant2
MEMCSn
Constant3
Relational1
data_rate
I_fb
Voltage gain:
Vin/500/5=Visgnal
changed 16 Aug 05
current gain:
Iin/5/10=Isignal
CMult1
ia
CMult2
ib
V_fb
EOCn2
EOCn
Delay3
pulse
AD_state1
FRSTDATAn1
Product
FRSTDATAn
Delay2
Concat
Mealy State Machine
Delay4
Subsystem
CMult3
v_ab
CMult4
v_bc
Step
Gain1
Gain2
Gain3
Gain4
Add
Add1
Multiport Switch
DATA_IN
Delay1
delay added for metastability
AD_state
inpin
Ready
100%
ode2

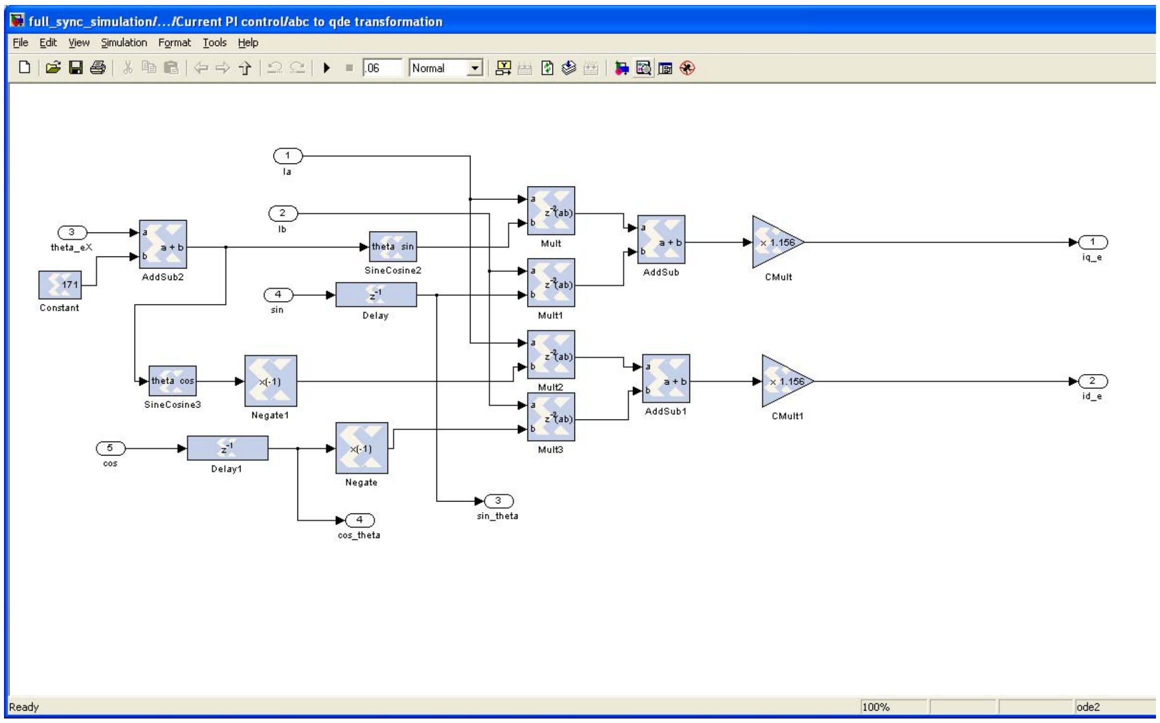

full_sync_simulation/Controller 1
File Edit View Simulation Format Tools Help
.06
Normal
5
fault_in
6
manual_fault
COMM2
Logical
CORDIC ATAN
Mult
Delay
modulation
vqd_mag
theta
SA
SB
SC
1
SA
2
SB
3
SC
4
fault_out
theta1
theta_XILINX
fault
sync out
5
theta_sync_out
7
Shut off
Logical1
Constant1
Constant2
vq_e_ref
vq_e
vd_e
vd_e_ref
reset
iq_e_ref
id_e_ref
iq_sync out
id_sync out
Voltage PI controller
1
vab
2
vbc
v_ab
v_bc
theta_eX
vq_e
vd_e
sine
cos
abc to qde transformation1
3
Ia
4
Ib
reset
theta
iqe_r
ide_r
sin
cos
Ia
Ib
vq_s_ref
vd_s_ref
vq sync out
vd sync out
Current PI control
In1
In2
In3
In4
Out1
Subsystem1
U45
6
integrator values
Ready
100%
ode2


full_sync_simulation/Controller 1/theta1
File Edit View Simulation Format Tools Help
.06
Normal
Angle_test_point
Relational2
U5
omega*delta_T
Accumulator
CMult
theta_XILINX
Two_PI
Relational
fault
Inverter7
Logical1
U44
sync out
Ready
100%
ode2

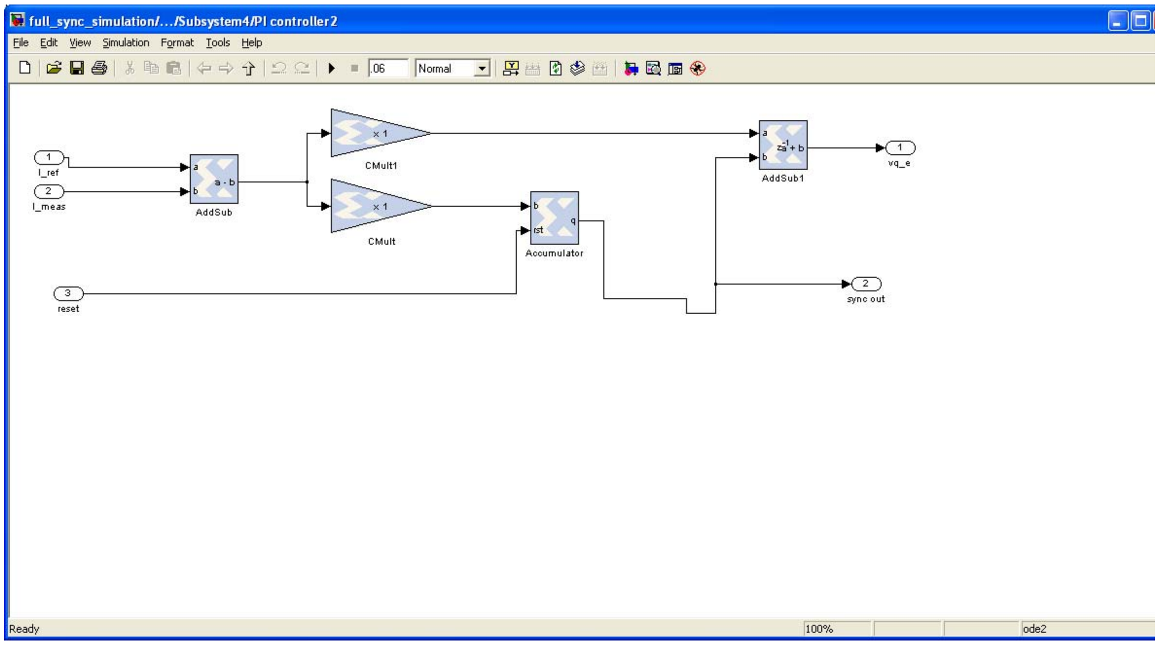

full_sync_simulation/Controller 1/abc to qde transformation1
File Edit View Simulation Format Tools Help
.06
Normal
1 v_ab
2 v_bc
3 theta_eX
Constant
AddSub2
SineCosine2
SineCosine3
Negate1
SineCosine1
Mult
Mult1
Mult2
Mult3
AddSub
AddSub1
CMult
CMult1
1 vq_e
2 vd_e
3 sine
4 cos
Ready
100%
ode2


full_sync_simulation/.../Voltage PI controller/PI controller2
File Edit View Simulation Format Tools Help
.06
Normal
1 vq_e_ref
2 v_meas
3 reset
AddSub
CMult1
CMult
Accumulator
AddSub1
1 iq_e_ref
2 sync out
Ready
100%
ode2

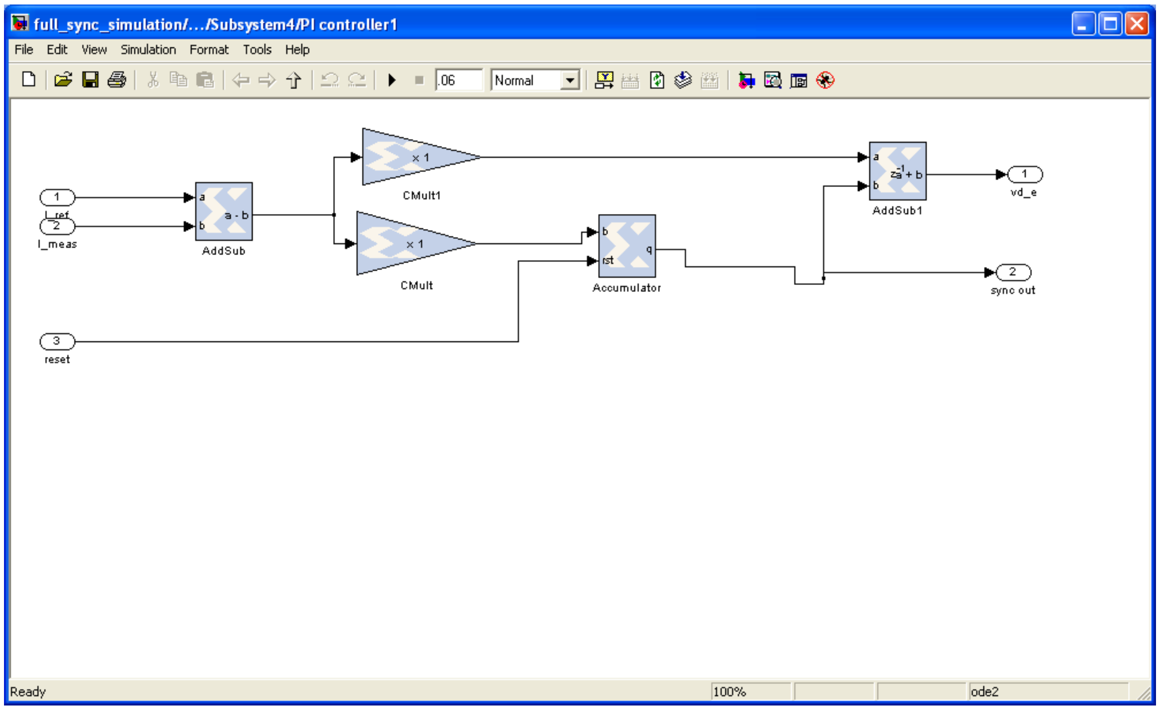

full_sync_simulation/.../Voltage PI controller/PI controller1
File Edit View Simulation Format Tools Help
.06
Normal
1 vd_e_ref
2 v_meas
3 reset
AddSub
CMult1
CMult
Accumulator
AddSub1
1 id_e_ref
2 sync out
Ready
100%
ode2

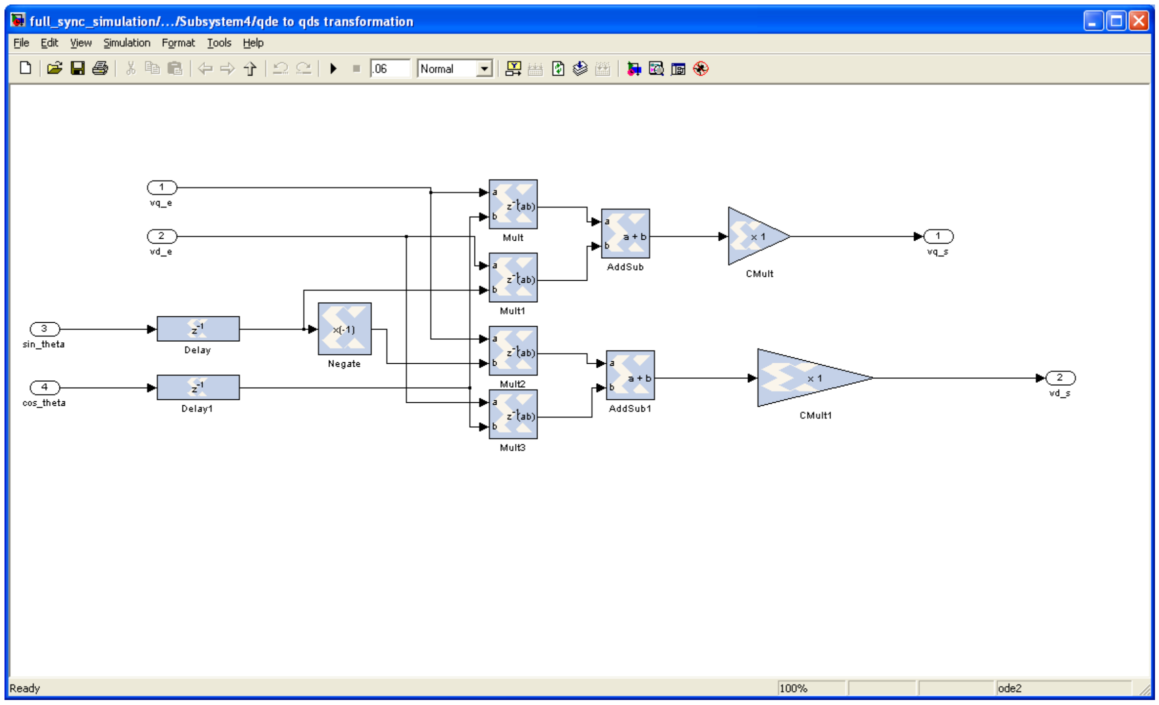


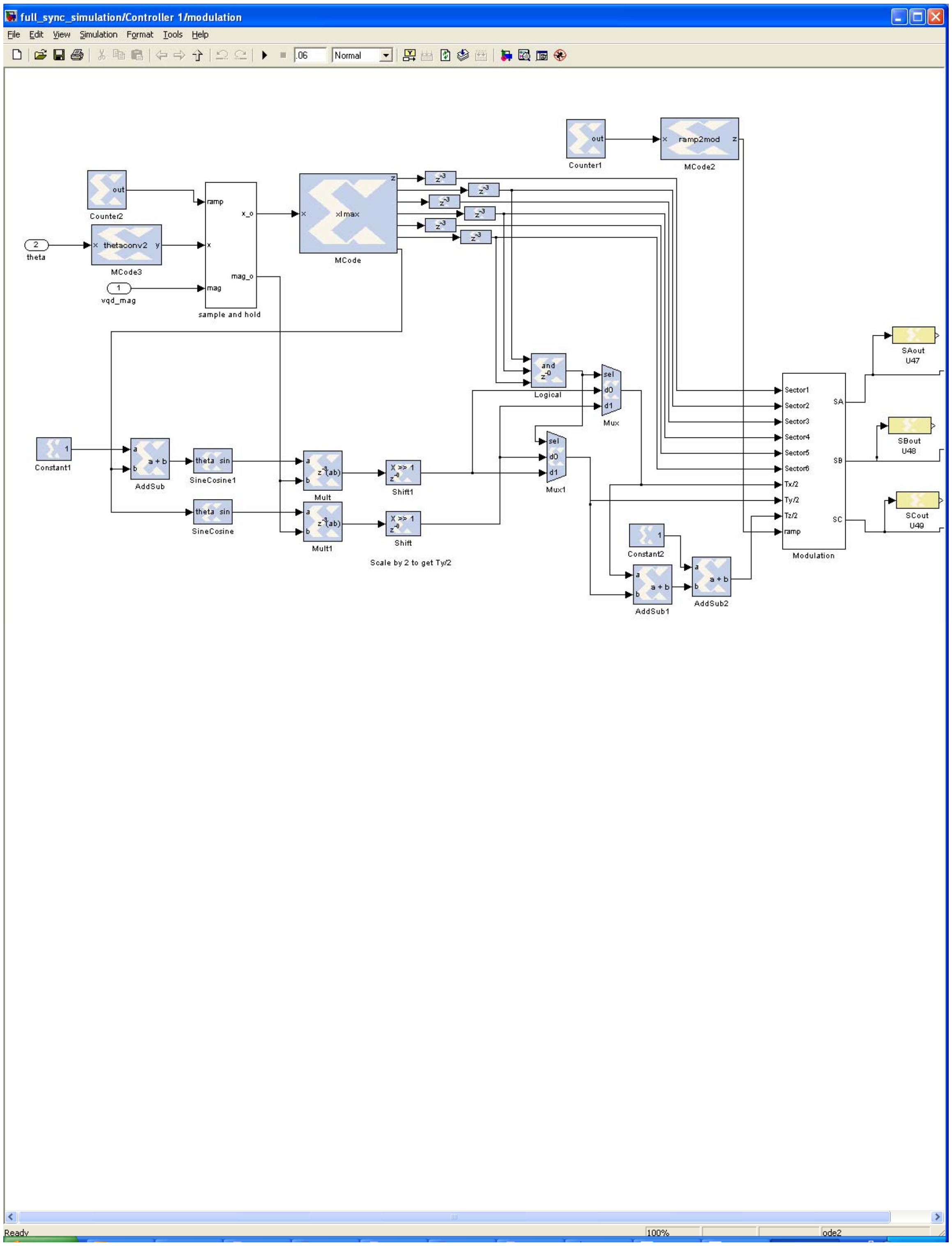
full_sync_simulation/.../Current PI control/abc to qde transformation
File Edit View Simulation Format Tools Help
.06
Normal
Ia
Ib
theta_eX
Constant
AddSub2
SineCosine2
sin
Delay
SineCosine3
Negate1
cos
Delay1
Negate
cos_theta
sin_theta
Mult
Mult1
Mult2
Mult3
AddSub
AddSub1
CMult
CMult1
iq_e
id_e
Ready
100%
ode2

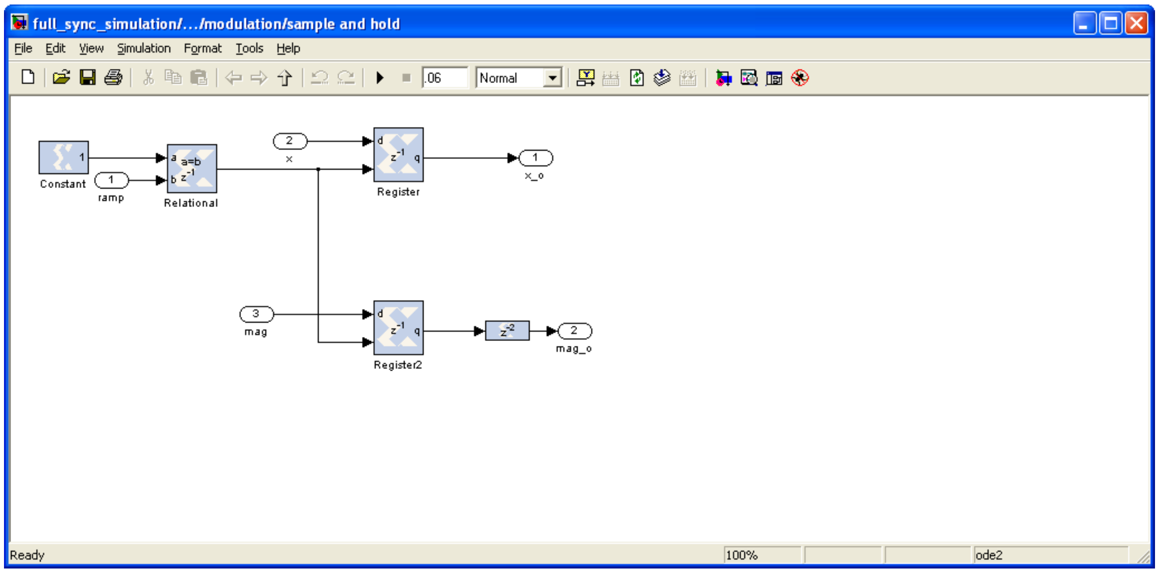


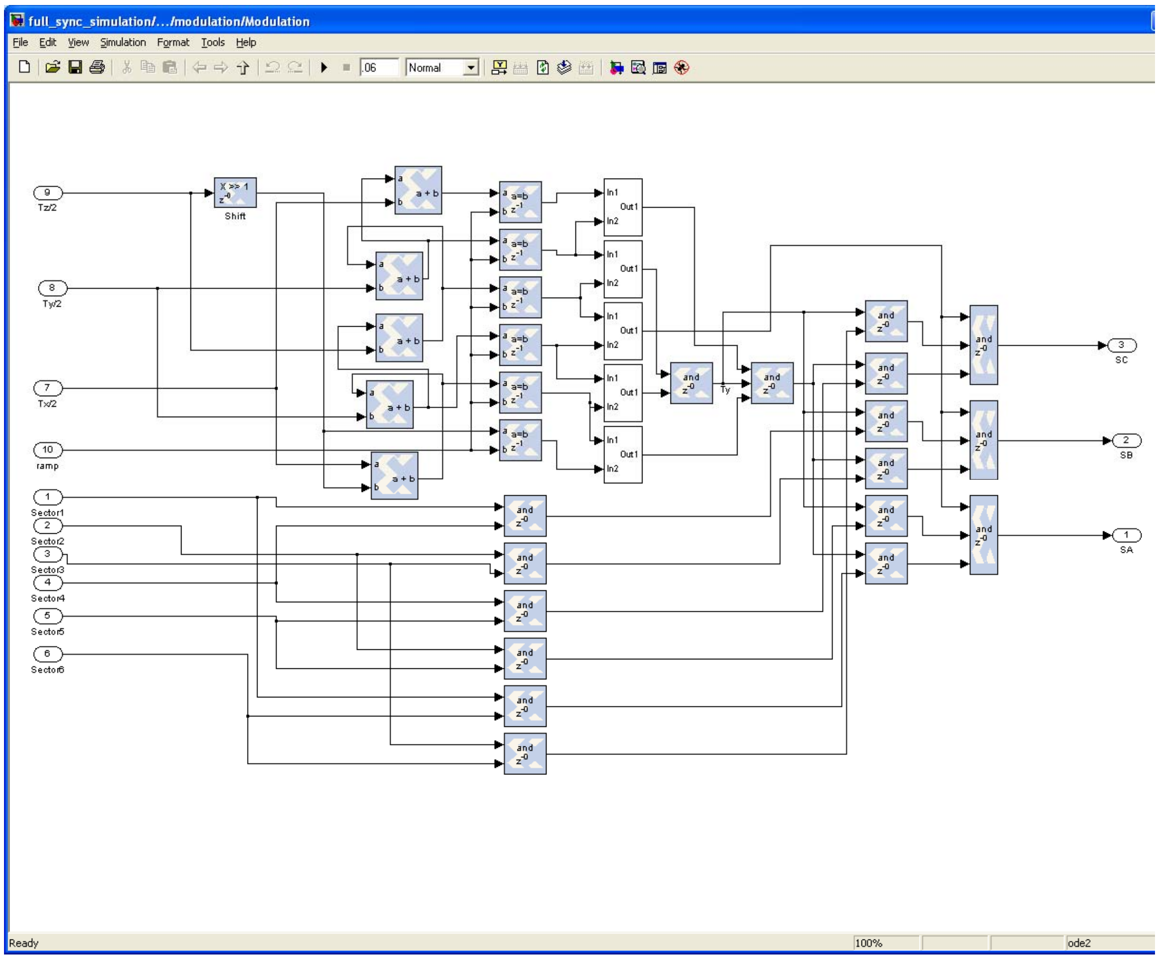
full_sync_simulation/.../Subsystem4/PI controller2
File Edit View Simulation Format Tools Help
.06
Normal
I_ref
I_meas
AddSub
CMult1
CMult
Accumulator
reset
AddSub1
vq_e
sync out
Ready
100%
ode2

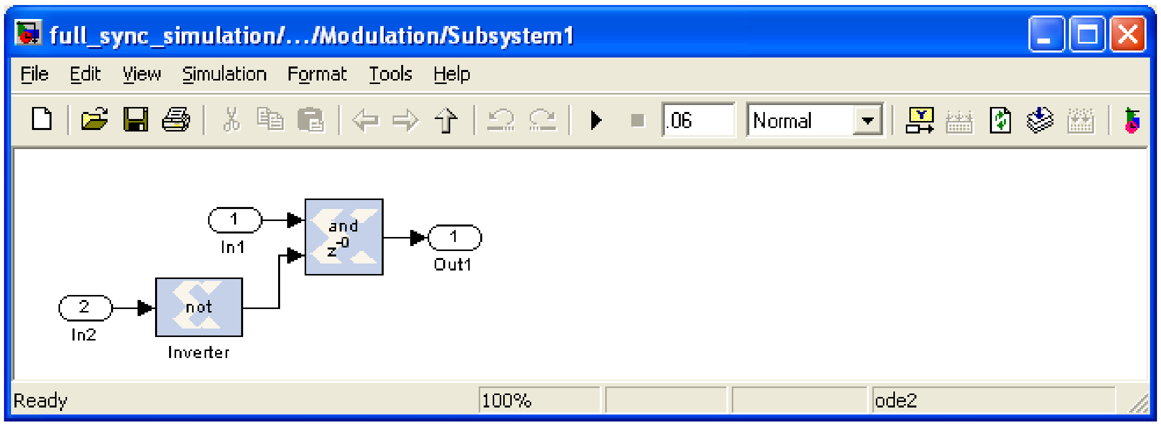

full_sync_simulation/.../Subsystem4/PI controller1
File Edit View Simulation Format Tools Help
.06
Normal
1
I_ref
2
I_meas
3
reset
AddSub
a - b
CMult1
× 1
CMult
× 1
Accumulator
rst
q
AddSub1
z⁻¹a + b
1
vd_e
2
sync out
Ready
100%
ode2

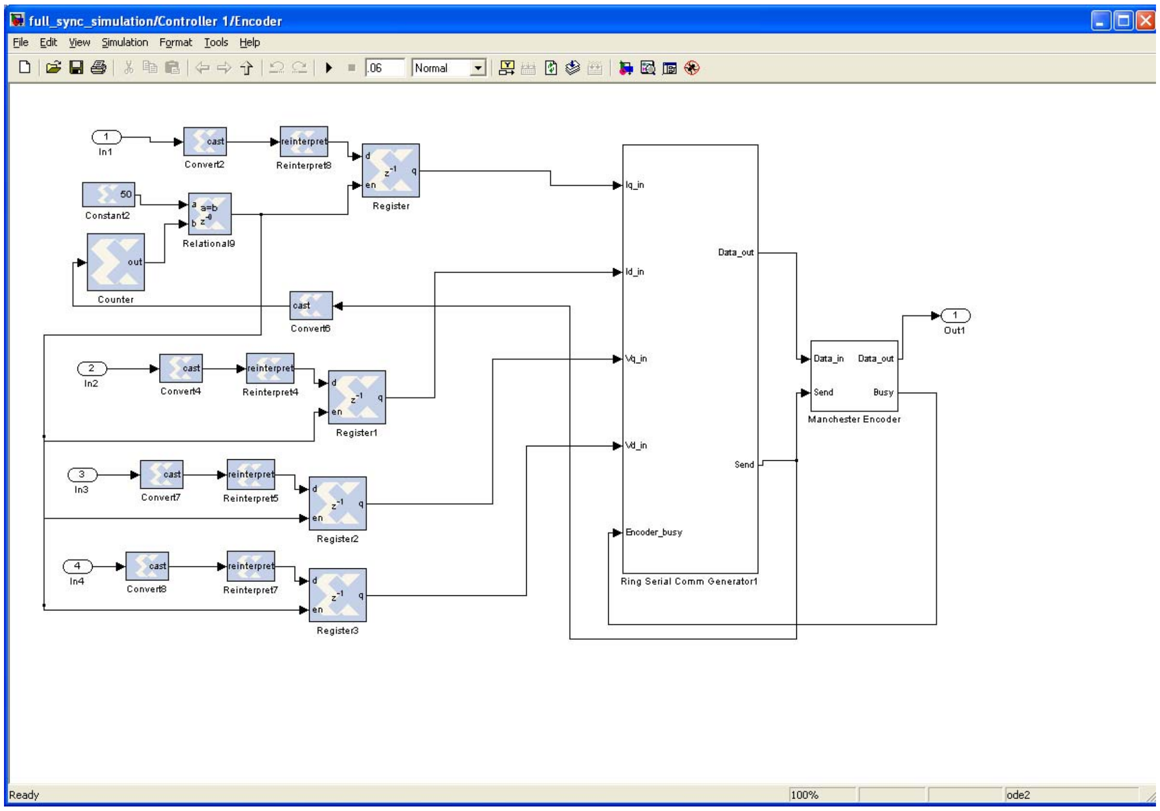

full_sync_simulation/.../Subsystem4/qde to qds transformation
File Edit View Simulation Format Tools Help
.06
Normal
1
vq_e
2
vd_e
3
sin_theta
4
cos_theta
Delay
Delay1
Negate
x(-1)
Mult
Mult1
Mult2
Mult3
z⁻¹(ab)
AddSub
AddSub1
a + b
CMult
CMult1
× 1
1
vq_s
2
vd_s
Ready
100%
ode2

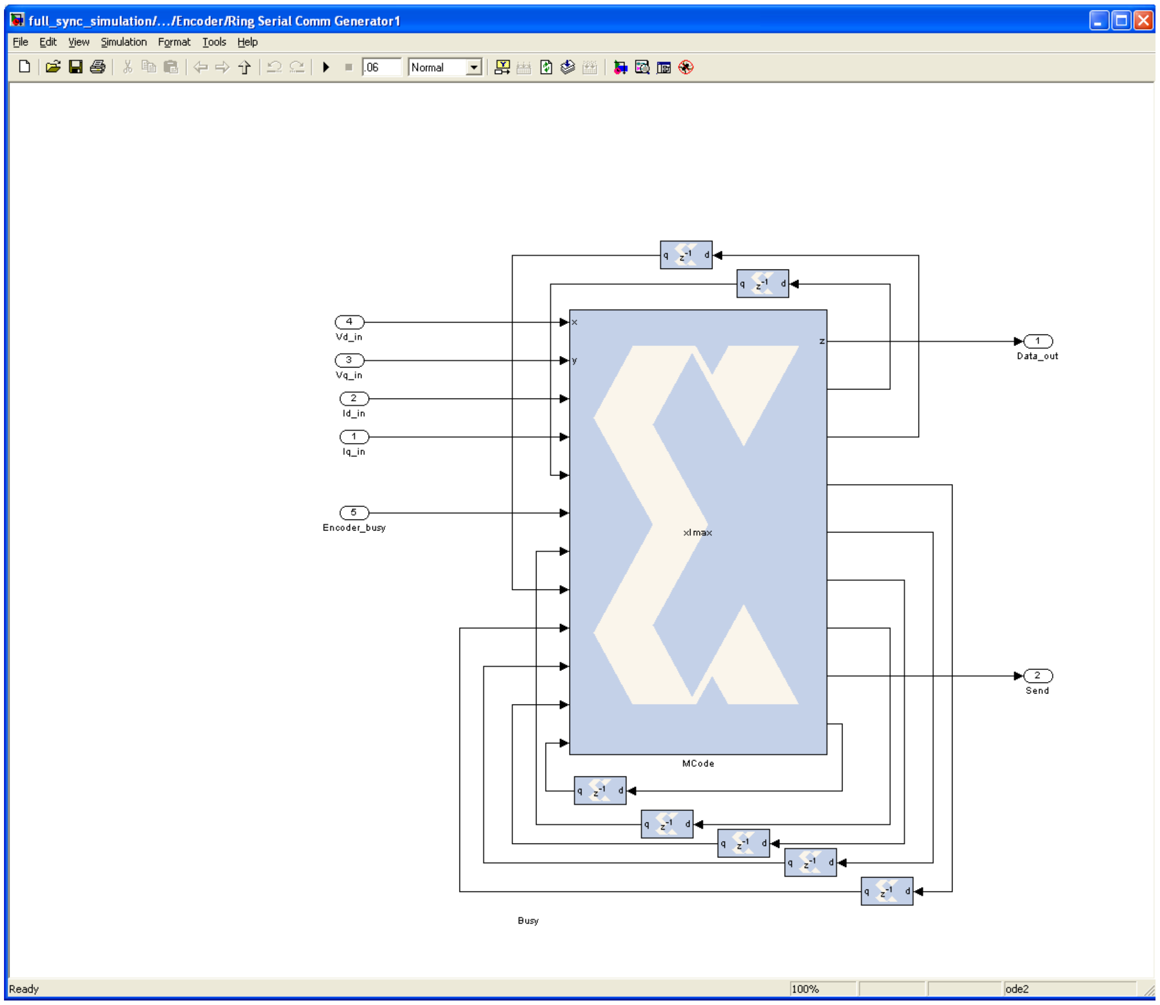

full_sync_simulation/Controller 1/modulation
File Edit View Simulation Format Tools Help
.06
Normal
Counter1
ramp2mod
MCode2
Counter2
theta
thetaconv2
MCode3
vqd_mag
sample and hold
xlmax
MCode
Logical
Mux
Mux1
Constant1
AddSub
SineCosine1
SineCosine
Mult
Mult1
Shift1
Shift
Scale by 2 to get Ty/2
Constant2
AddSub1
AddSub2
Modulation
Sector1
Sector2
Sector3
Sector4
Sector5
Sector6
Tx/2
Ty/2
Tz/2
ramp
SA
SB
SC
SAout U47
SBout U48
SCout U49
Ready
100%
ode2

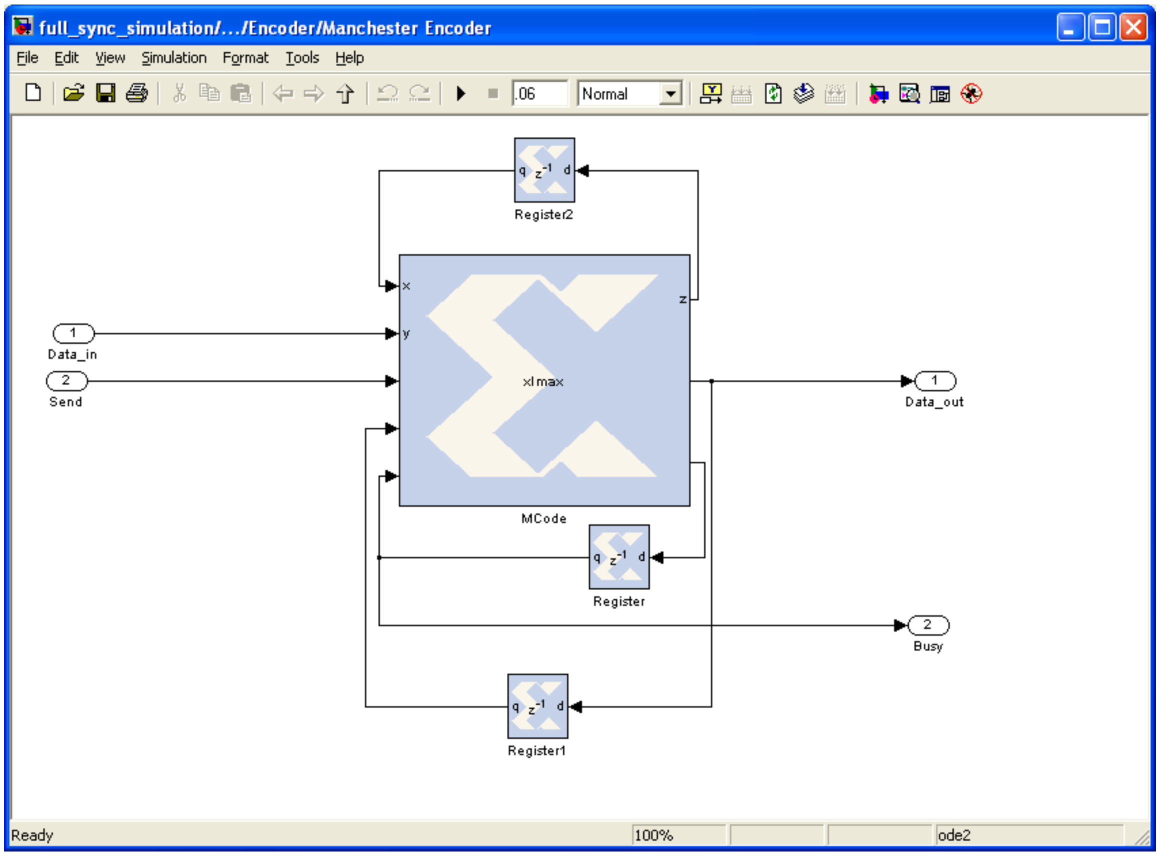

full_sync_simulation/.../modulation/sample and hold
File Edit View Simulation Format Tools Help
.06
Normal
Constant
ramp
Relational
x
Register
x_o
mag
Register2
mag_o
Ready
100%
ode2

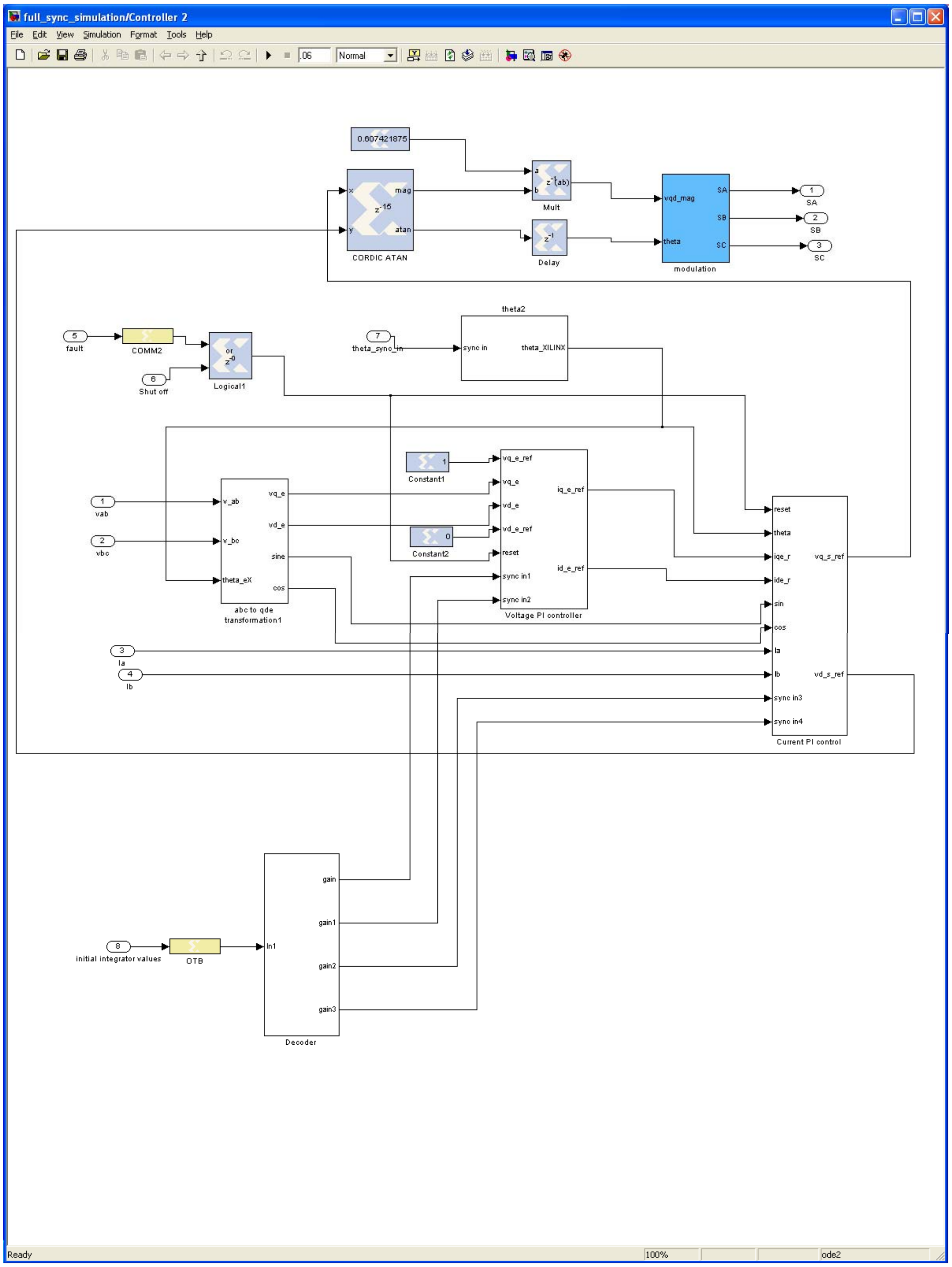

full_sync_simulation/.../modulation/Modulation
File Edit View Simulation Format Tools Help
.06
Normal
Tz/2
Shift
Ty/2
Tx/2
ramp
Sector1
Sector2
Sector3
Sector4
Sector5
Sector6
SC
SB
SA
Ready
100%
ode2


full_sync_simulation/.../Modulation/Subsystem1
File Edit View Simulation Format Tools Help
.06
Normal
1
In1
and
z-0
1
Out1
2
In2
not
Inverter
Ready
100%
ode2

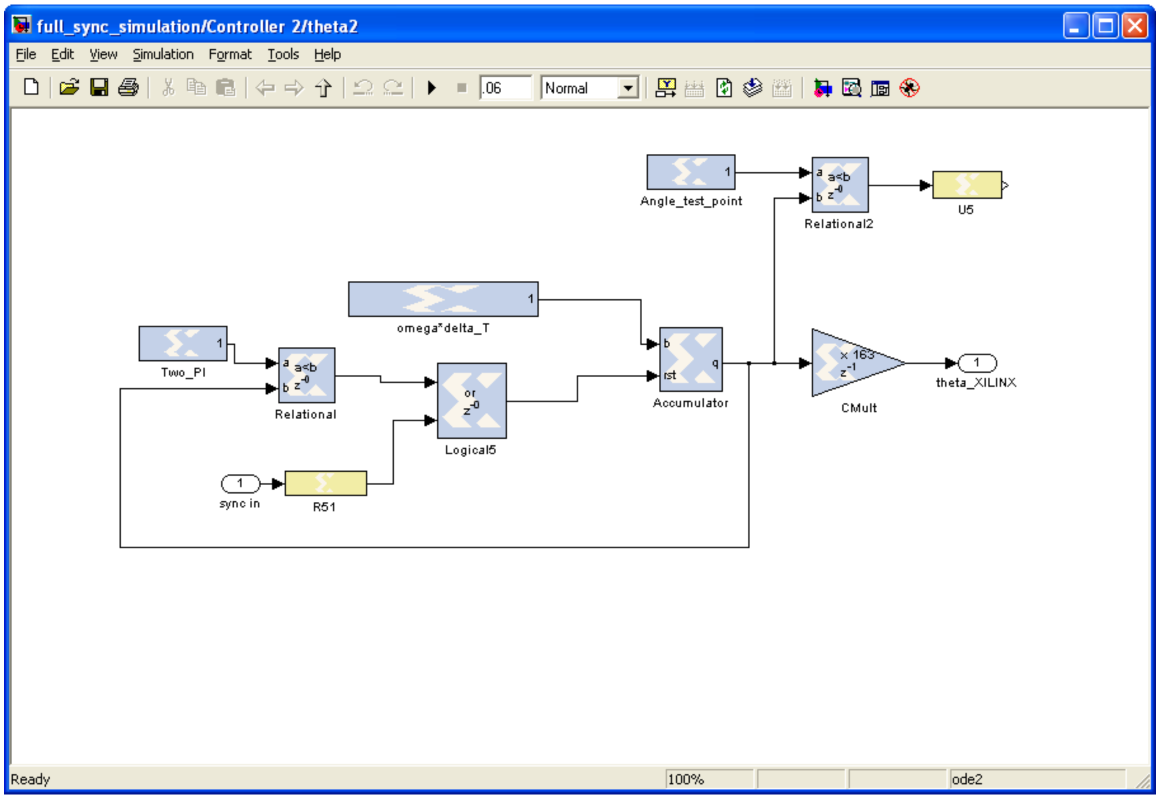

full_sync_simulation/Controller 1/Encoder
File Edit View Simulation Format Tools Help
.06
Normal
1
In1
cast
Convert2
reinterpret
Reinterpret8
d
z-1
q
en
Register
50
Constant2
a=b
z-0
Relational9
out
Counter
cast
Convert6
2
In2
cast
Convert4
reinterpret
Reinterpret4
Register1
3
In3
cast
Convert7
reinterpret
Reinterpret5
Register2
4
In4
cast
Convert8
reinterpret
Reinterpret7
Register3
Iq_in
Id_in
Vq_in
Vd_in
Encoder_busy
Data_out
Send
Ring Serial Comm Generator1
Data_in
Data_out
Send
Busy
Manchester Encoder
1
Out1
Ready
100%
ode2

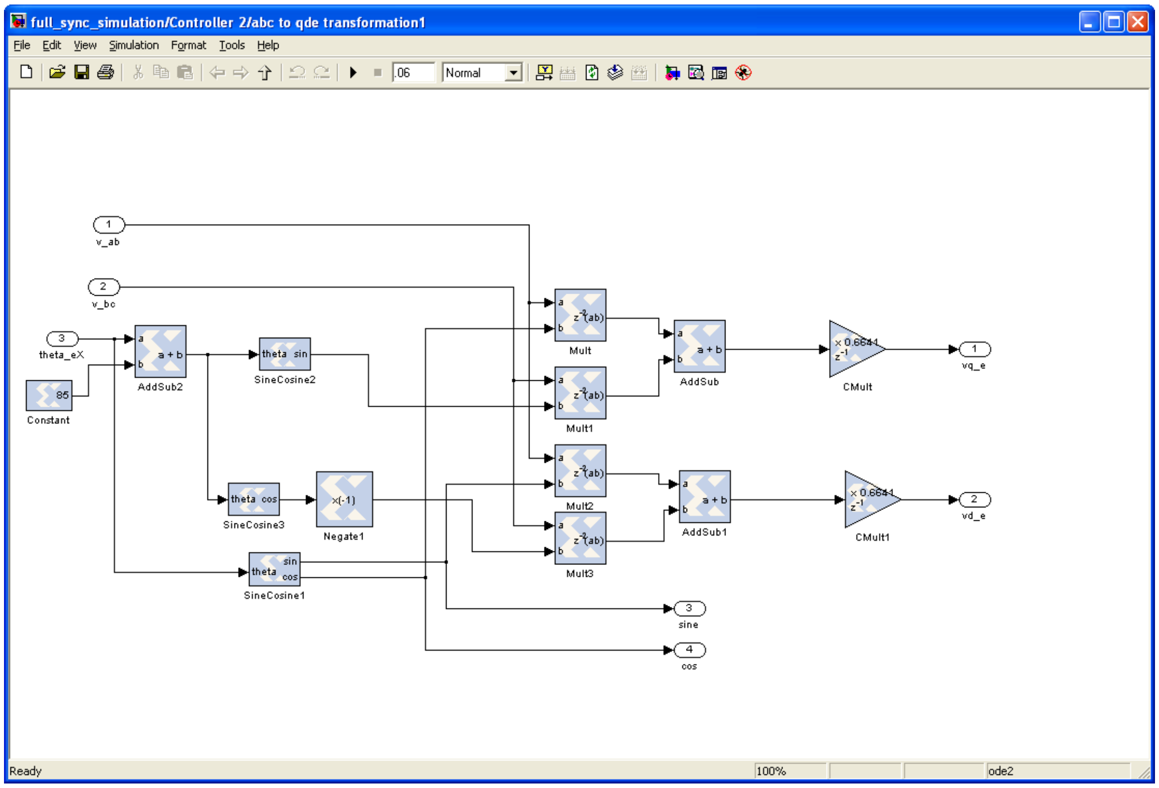

full_sync_simulation/.../Encoder/Ring Serial Comm Generator1
File Edit View Simulation Format Tools Help
Normal
4
Vd_in
3
Vq_in
2
Id_in
1
Iq_in
5
Encoder_busy
x
y
z
xlmax
MCode
1
Data_out
2
Send
q z-1 d
Busy
Ready
100%
ode2

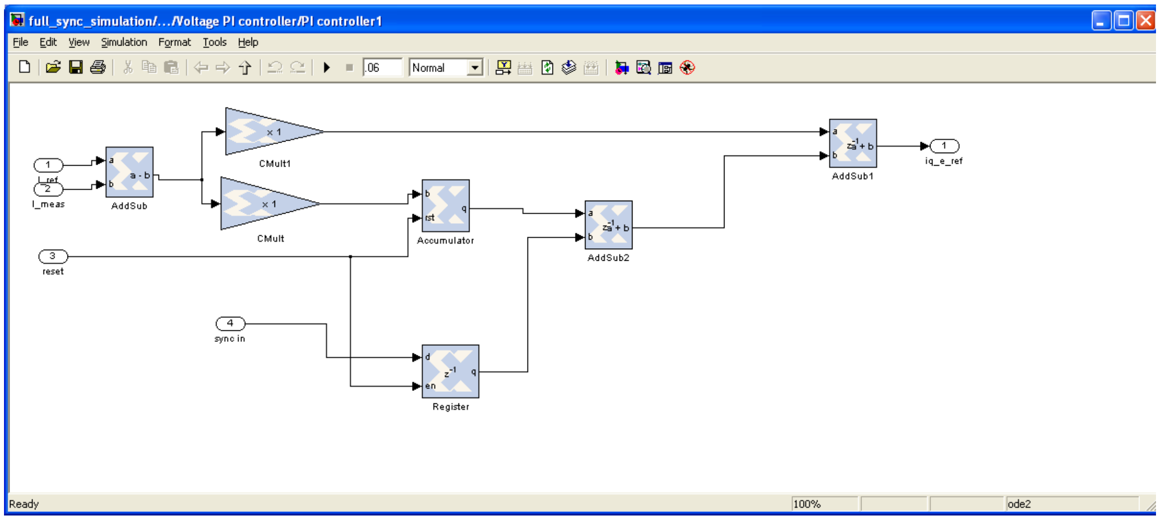

full_sync_simulation/.../Encoder/Manchester Encoder
File Edit View Simulation Format Tools Help
.06
Normal
Register2
x
y
z
xlmax
MCode
1
Data_in
2
Send
1
Data_out
Register
2
Busy
Register1
q
z-1
d
Ready
100%
ode2

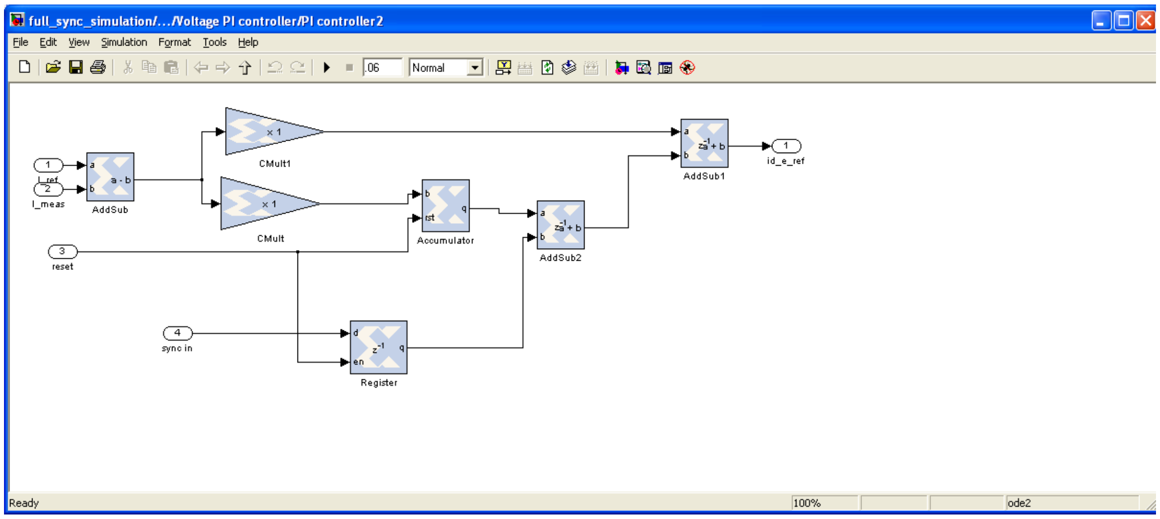


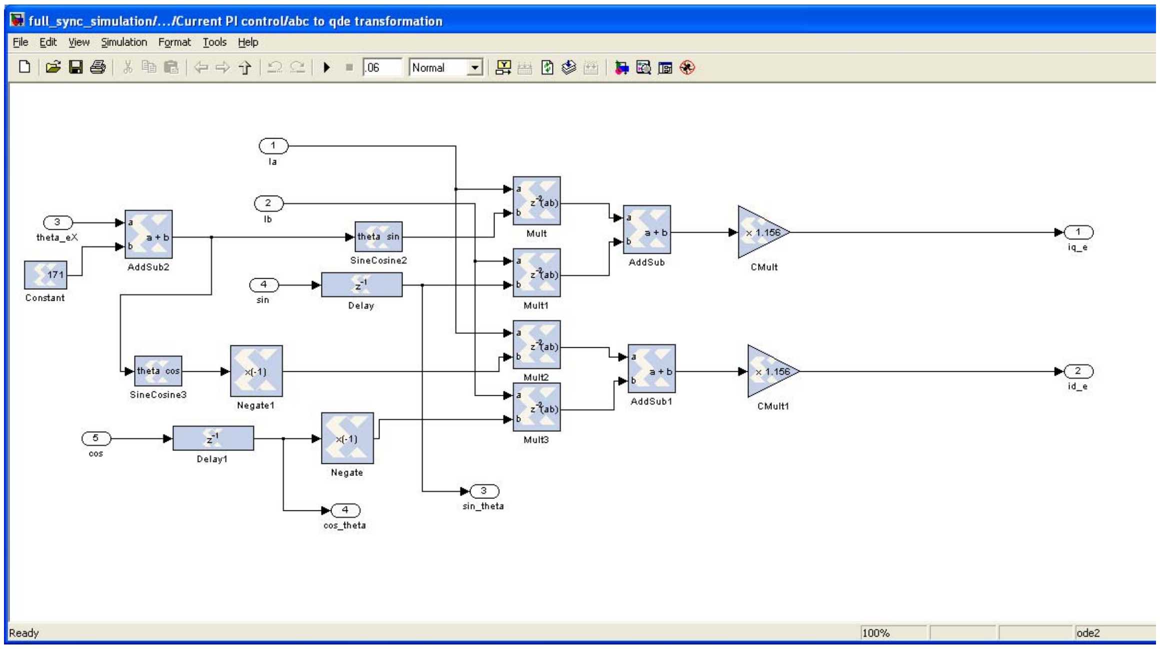
full_sync_simulation/Controller 2
File Edit View Simulation Format Tools Help
.06
Normal
0.607421875
CORDIC ATAN
Mult
Delay
modulation
vqd_mag
theta
SA
SB
SC
fault
COMM2
Shut off
Logical1
theta_sync_in
theta2
sync in
theta_XILINX
Constant1
Constant2
vab
vbc
abc to qde transformation1
Voltage PI controller
Current PI control
Ia
Ib
initial integrator values
OTB
Decoder
gain
gain1
gain2
gain3
Ready
100%
ode2

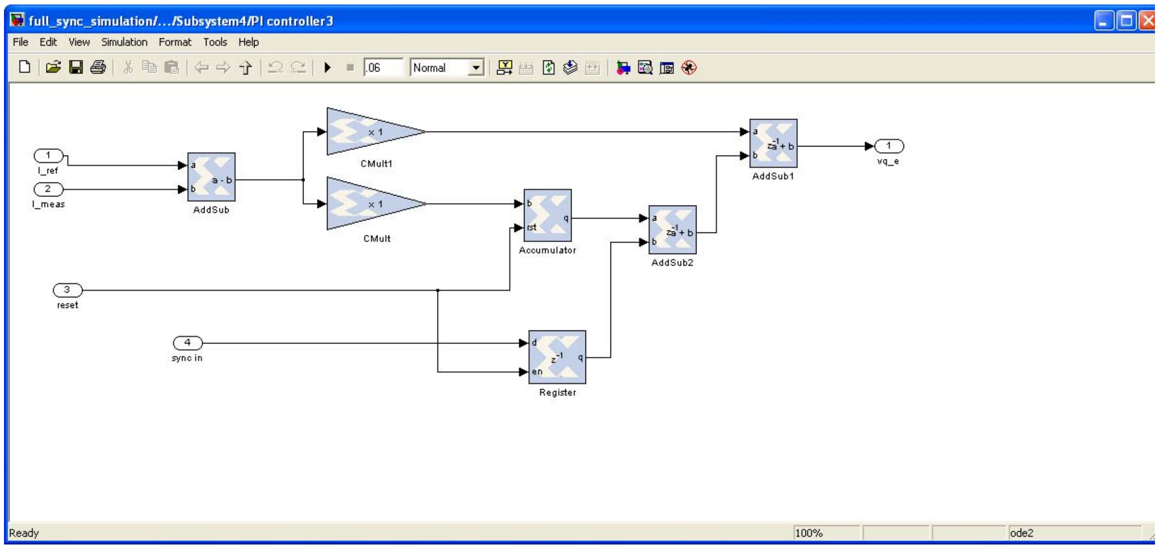

full_sync_simulation/Controller 2/theta2
File Edit View Simulation Format Tools Help
.06
Normal
Angle_test_point
Relational2
U5
omega*delta_T
Two_PI
Relational
Logical5
Accumulator
CMult
theta_XILINX
sync in
R51
Ready
100%
ode2

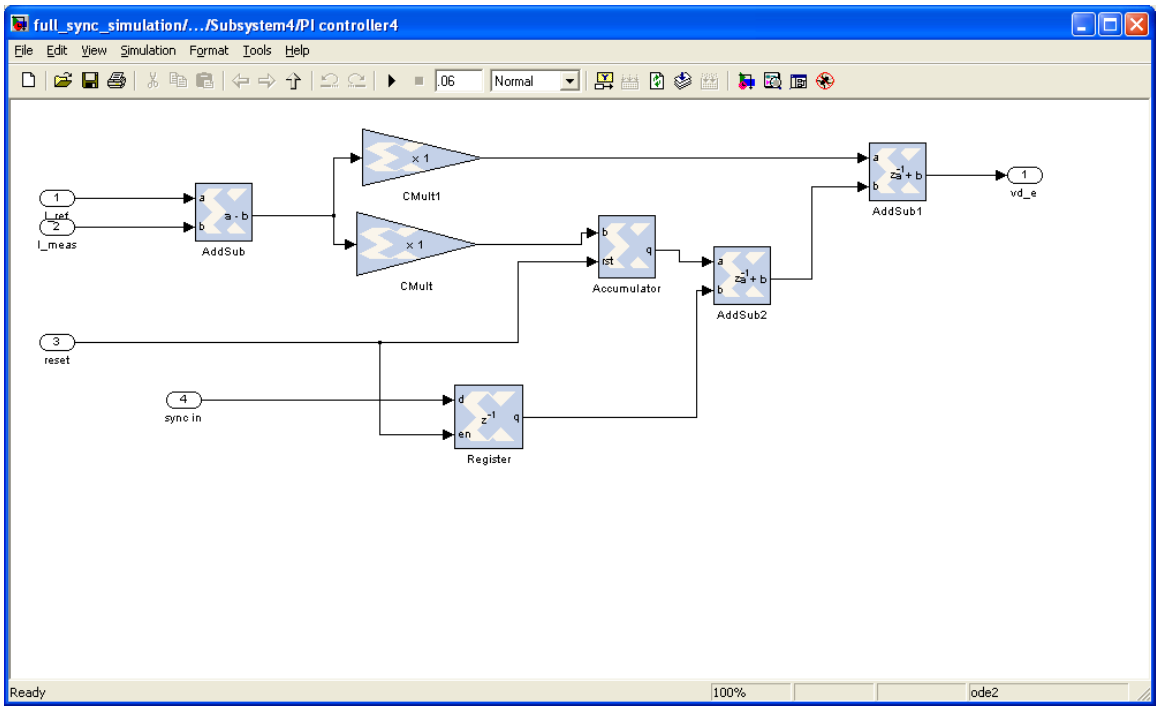

full_sync_simulation/Controller 2/abc to qde transformation1
File Edit View Simulation Format Tools Help
.06
Normal
v_ab
v_bc
theta_eX
Constant
AddSub2
SineCosine2
Mult
Mult1
AddSub
CMult
vq_e
SineCosine3
Negate1
Mult2
Mult3
AddSub1
CMult1
vd_e
SineCosine1
sine
cos
Ready
100%
ode2

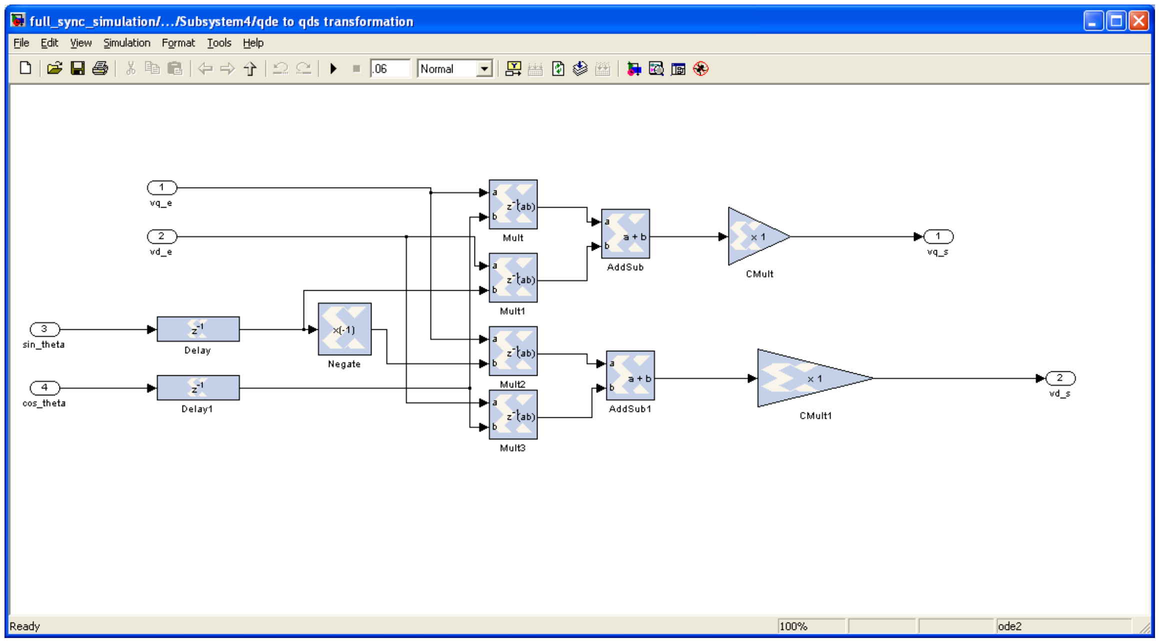

full_sync_simulation/.../Voltage PI controller/PI controller1
File Edit View Simulation Format Tools Help
.06
Normal
1
I_ref
2
I_meas
a - b
AddSub
x 1
CMult1
x 1
CMult
b
rst
q
Accumulator
a
b
z^-1a + b
AddSub2
AddSub1
1
iq_e_ref
3
reset
4
sync in
d
en
z^-1
q
Register
Ready
100%
ode2

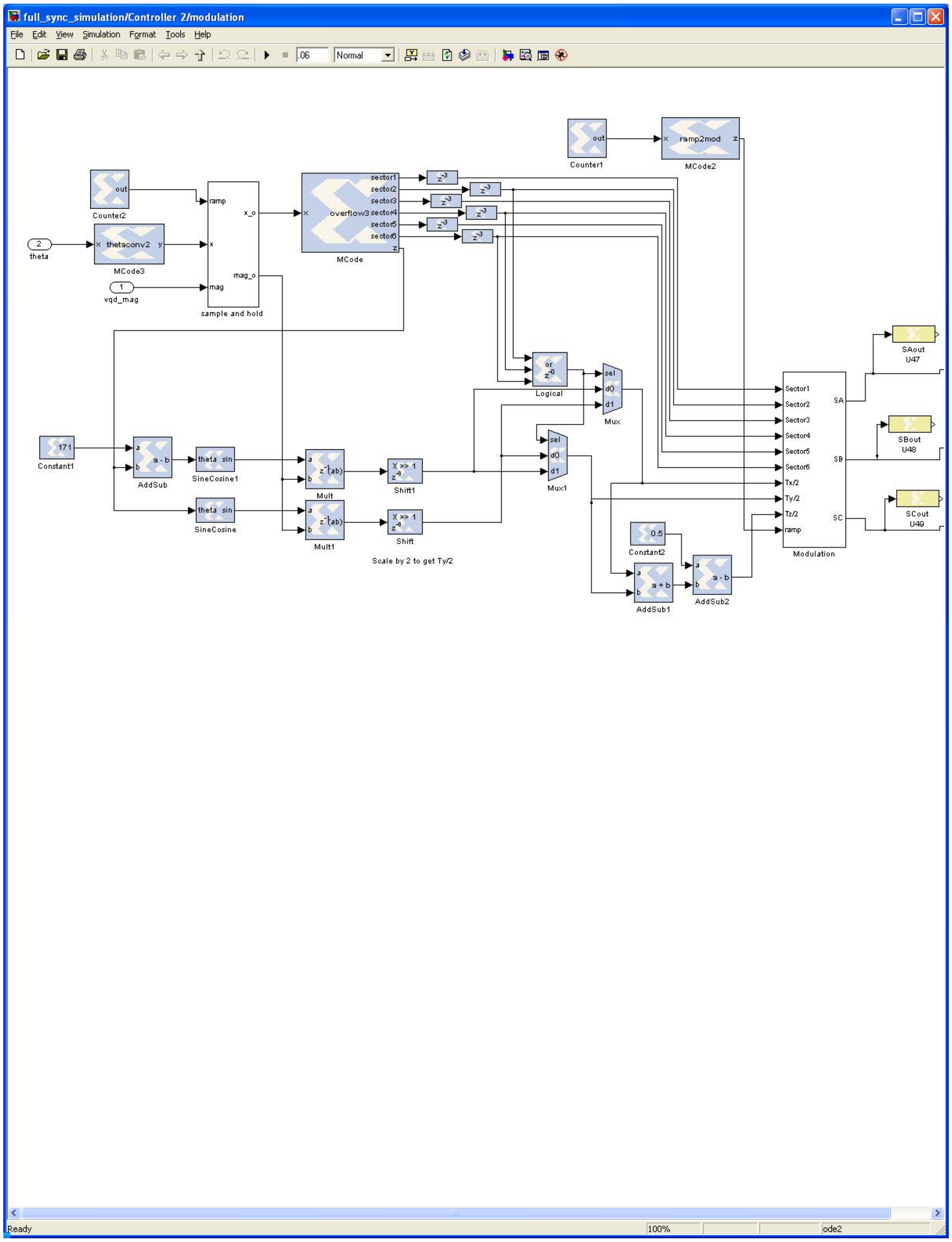

full_sync_simulation/.../Voltage PI controller/PI controller2
File Edit View Simulation Format Tools Help
.06
Normal
1
I_ref
2
I_meas
a - b
AddSub
x 1
CMult1
x 1
CMult
b
rst
q
Accumulator
a
b
z^-1a + b
AddSub2
AddSub1
1
id_e_ref
3
reset
4
sync in
d
en
z^-1
q
Register
Ready
100%
ode2

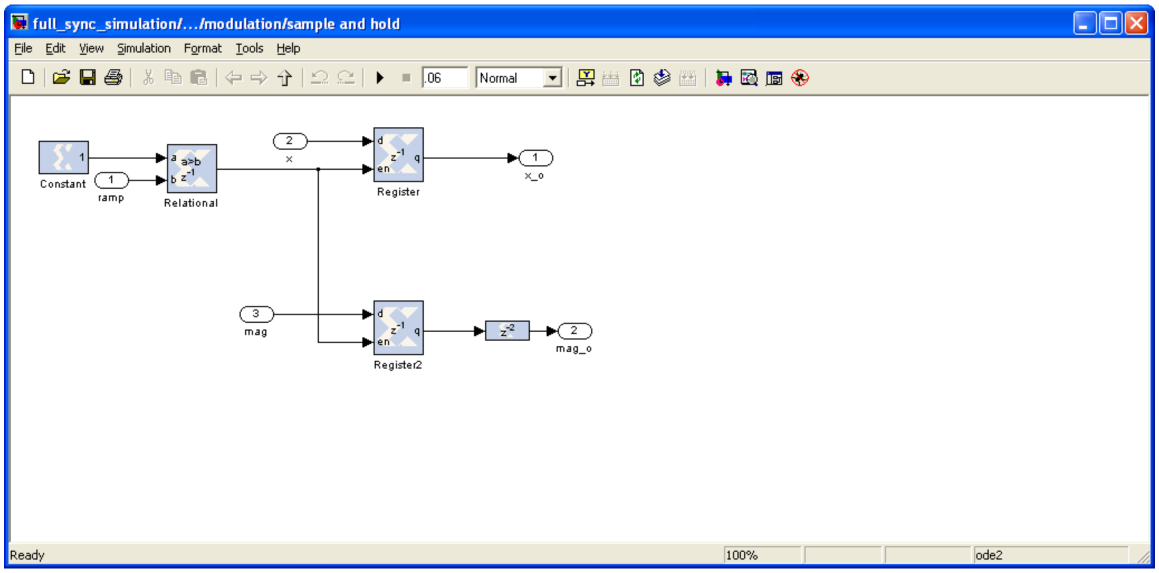

full_sync_simulation/.../Current PI control/abc to qde transformation
File Edit View Simulation Format Tools Help
.06
Normal
1
Ia
2
Ib
3
theta_eX
171
Constant
a + b
AddSub2
theta sin
SineCosine2
4
sin
z-1
Delay
z-3(ab)
Mult
z-3(ab)
Mult1
a + b
AddSub
× 1.156
CMult
1
iq_e
theta cos
SineCosine3
×(-1)
Negate1
z-3(ab)
Mult2
z-3(ab)
Mult3
a + b
AddSub1
× 1.156
CMult1
2
id_e
5
cos
z-1
Delay1
×(-1)
Negate
3
sin_theta
4
cos_theta
Ready
100%
ode2

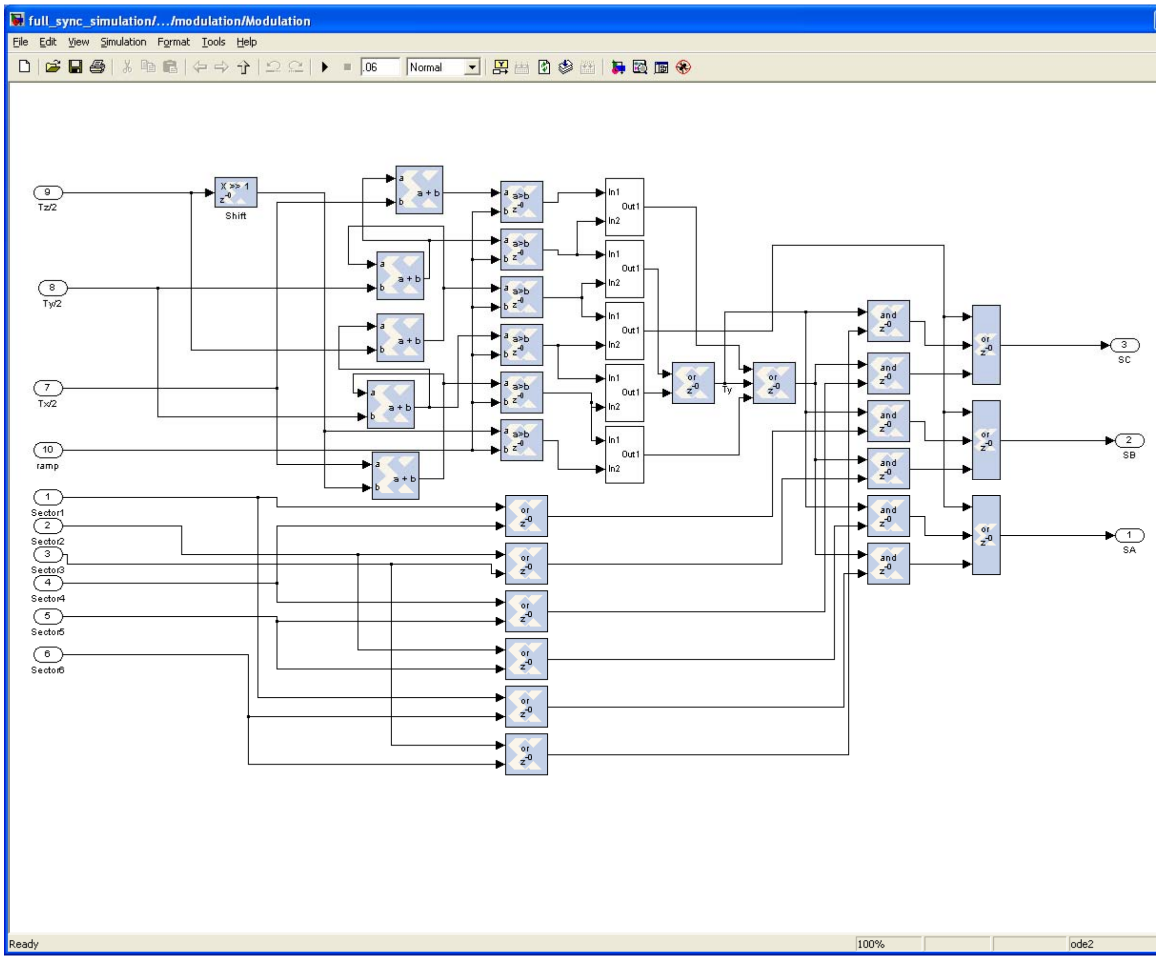

full_sync_simulation/.../Subsystem4/PI controller3
File Edit View Simulation Format Tools Help
.06
Normal
1
I_ref
2
I_meas
a - b
AddSub
× 1
CMult1
× 1
CMult
b
rst
q
Accumulator
za-1 + b
AddSub2
za-1 + b
AddSub1
1
vq_e
3
reset
4
sync in
d
en
z-1
q
Register
Ready
100%
ode2

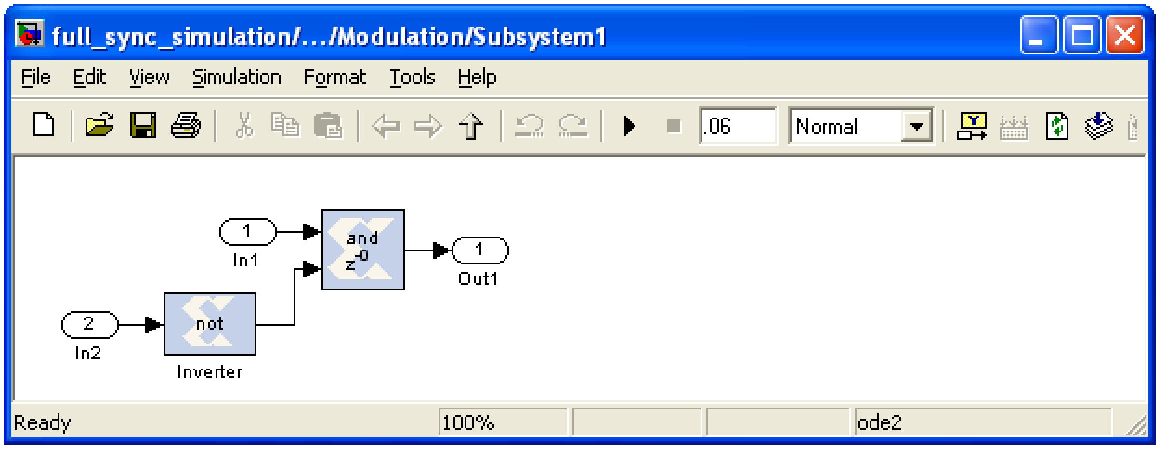

full_sync_simulation/.../Subsystem4/PI controller4
File Edit View Simulation Format Tools Help
.06
Normal
I_ref
I_meas
reset
sync in
AddSub
CMult1
CMult
Accumulator
AddSub2
AddSub1
Register
vd_e
Ready
100%
ode2

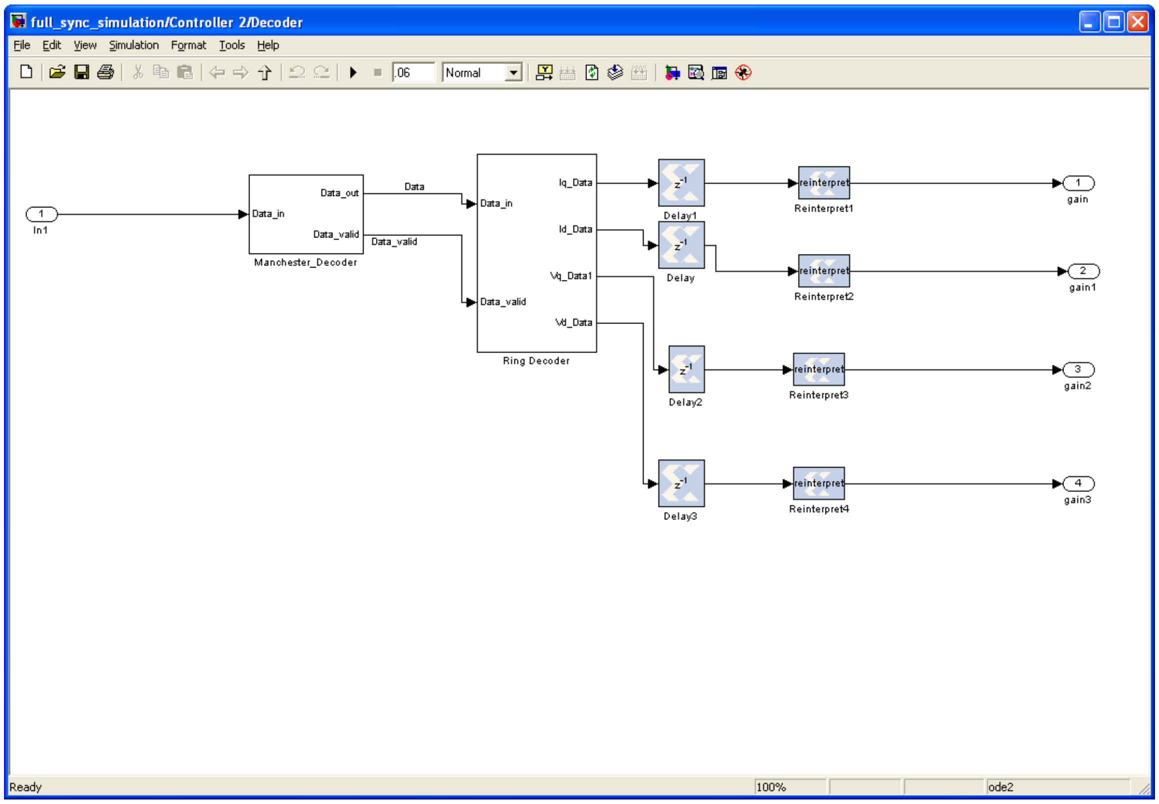

full_sync_simulation/.../Subsystem4/qde to qds transformation
File Edit View Simulation Format Tools Help
.06
Normal
vq_e
vd_e
sin_theta
cos_theta
Delay
Delay1
Negate
Mult
Mult1
Mult2
Mult3
AddSub
AddSub1
CMult
CMult1
vq_s
vd_s
Ready
100%
ode2

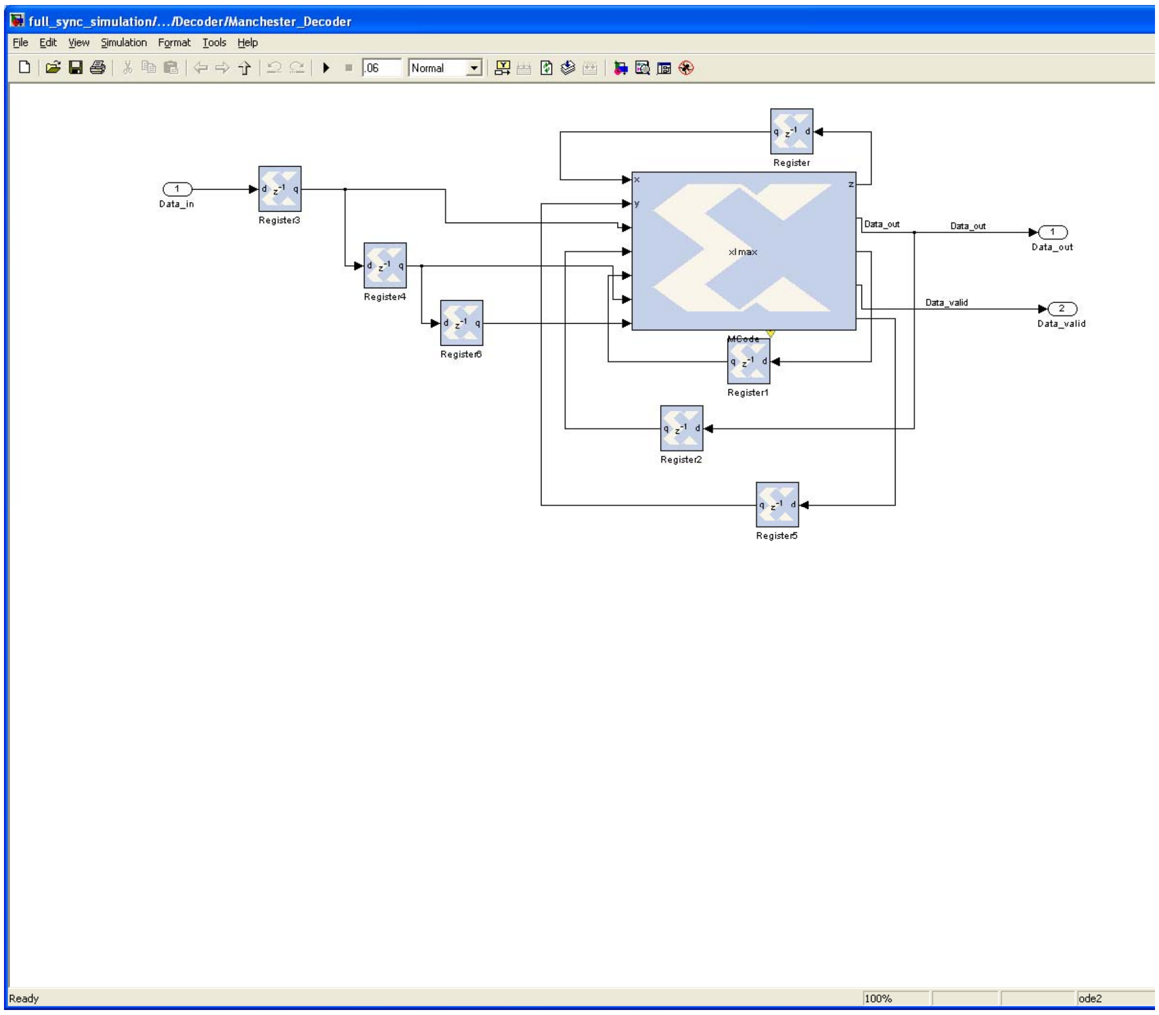

full_sync_simulation/Controller 2/modulation
File Edit View Simulation Format Tools Help
Normal
Counter1
MCode2
ramp2mod
Counter2
MCode3
thetaconv2
theta
vqd_mag
sample and hold
MCode
overflow3
sector1
sector2
sector3
sector4
sector5
sector6
Logical
Mux
Mux1
Constant1
AddSub
SineCosine1
SineCosine
Mult
Mult1
Shift1
Shift
Scale by 2 to get Ty/2
Constant2
AddSub1
AddSub2
Modulation
Sector1
Sector2
Sector3
Sector4
Sector5
Sector6
Tx/2
Ty/2
Tz/2
ramp
SA
SB
SC
SAout U47
SBout U48
SCout U49
Ready
100%
ode2

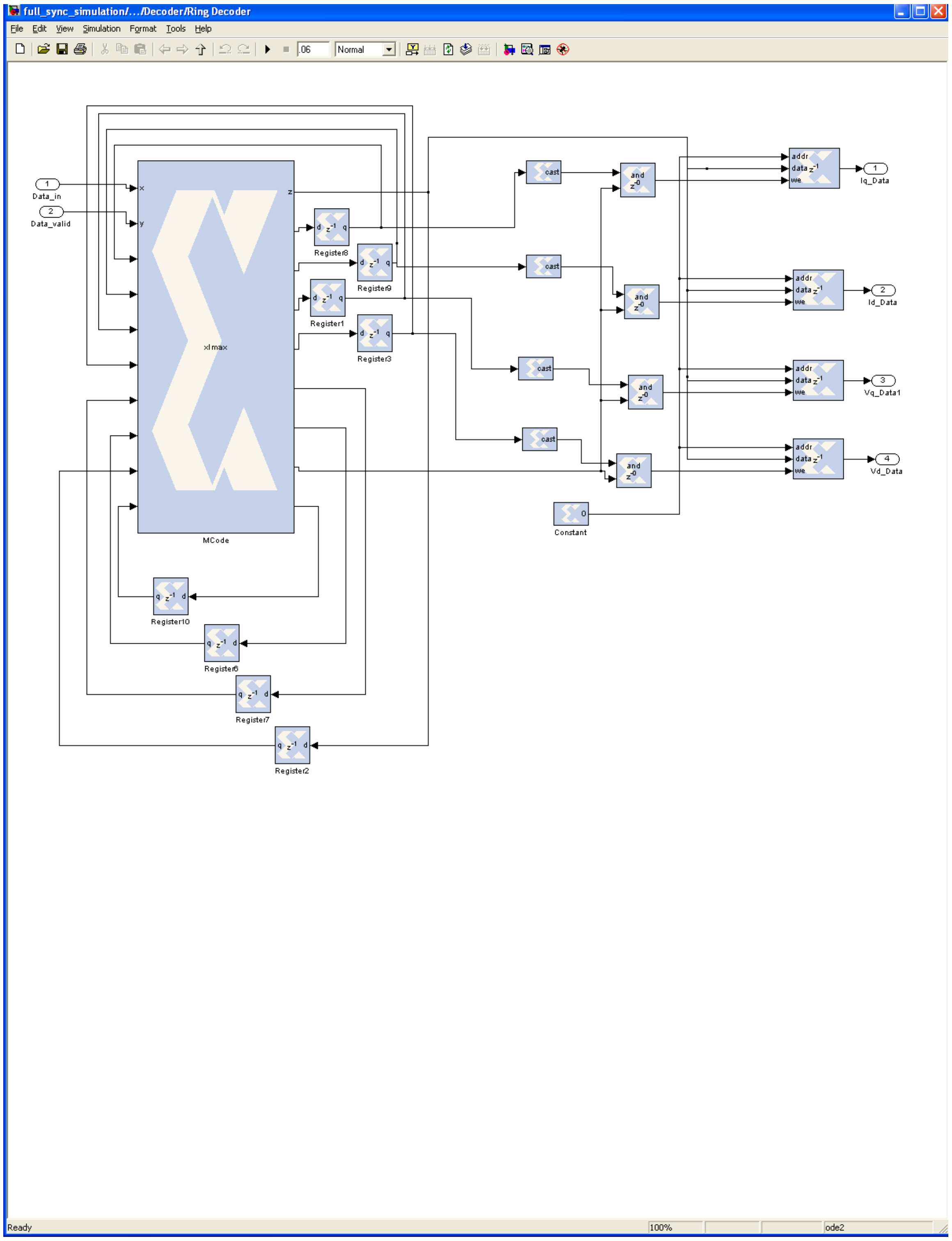

full_sync_simulation/.../modulation/sample and hold
File Edit View Simulation Format Tools Help
.06
Normal
Constant
ramp
Relational
x
Register
x_o
mag
Register2
mag_o
Ready
100%
ode2

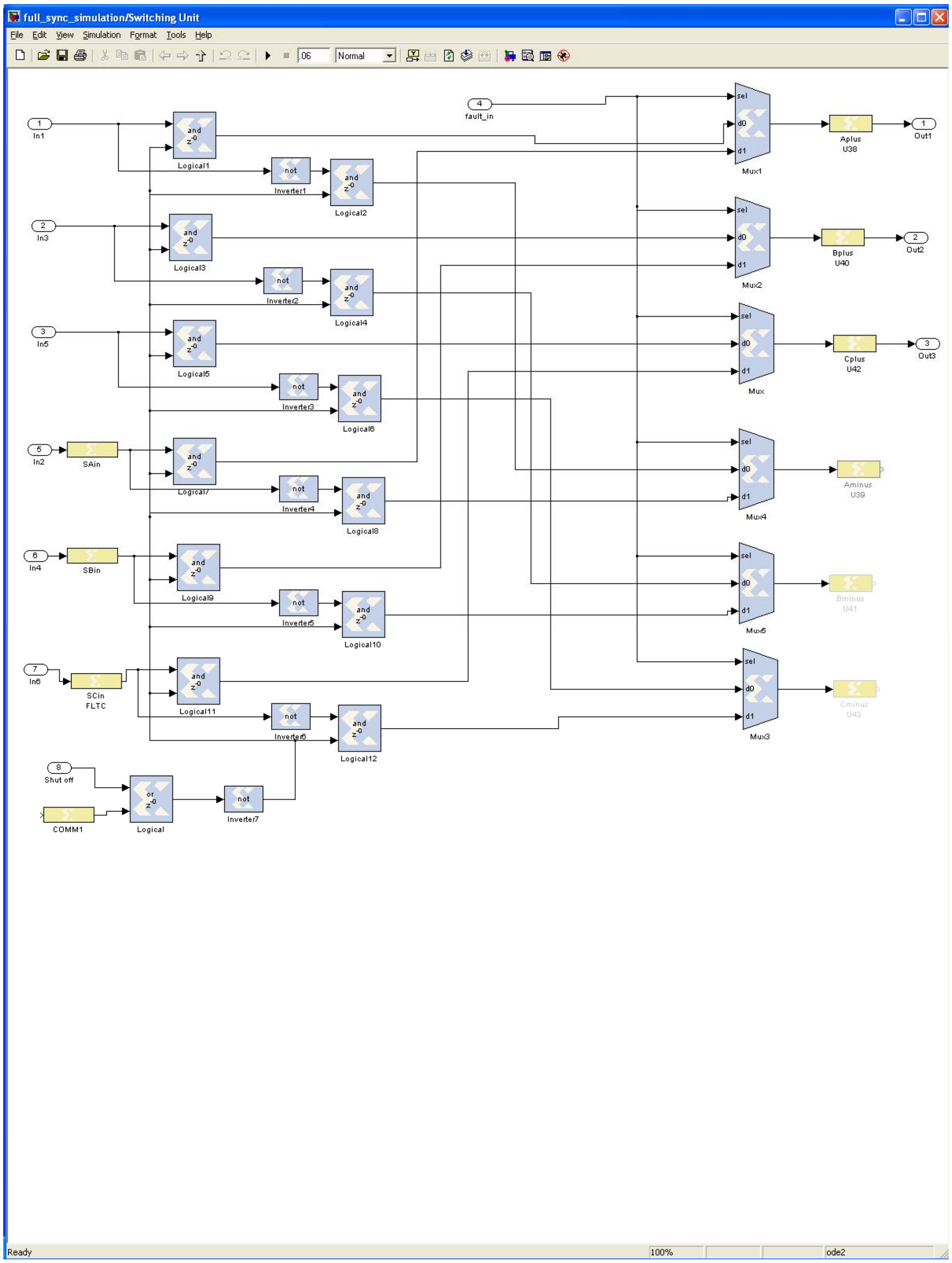

full_sync_simulation/.../modulation/Modulation
File Edit View Simulation Format Tools Help
.06
Normal
Tz/2
Shift
Ty/2
Tx/2
ramp
Sector1
Sector2
Sector3
Sector4
Sector5
Sector6
Ty
SC
SB
SA
Ready
100%
ode2

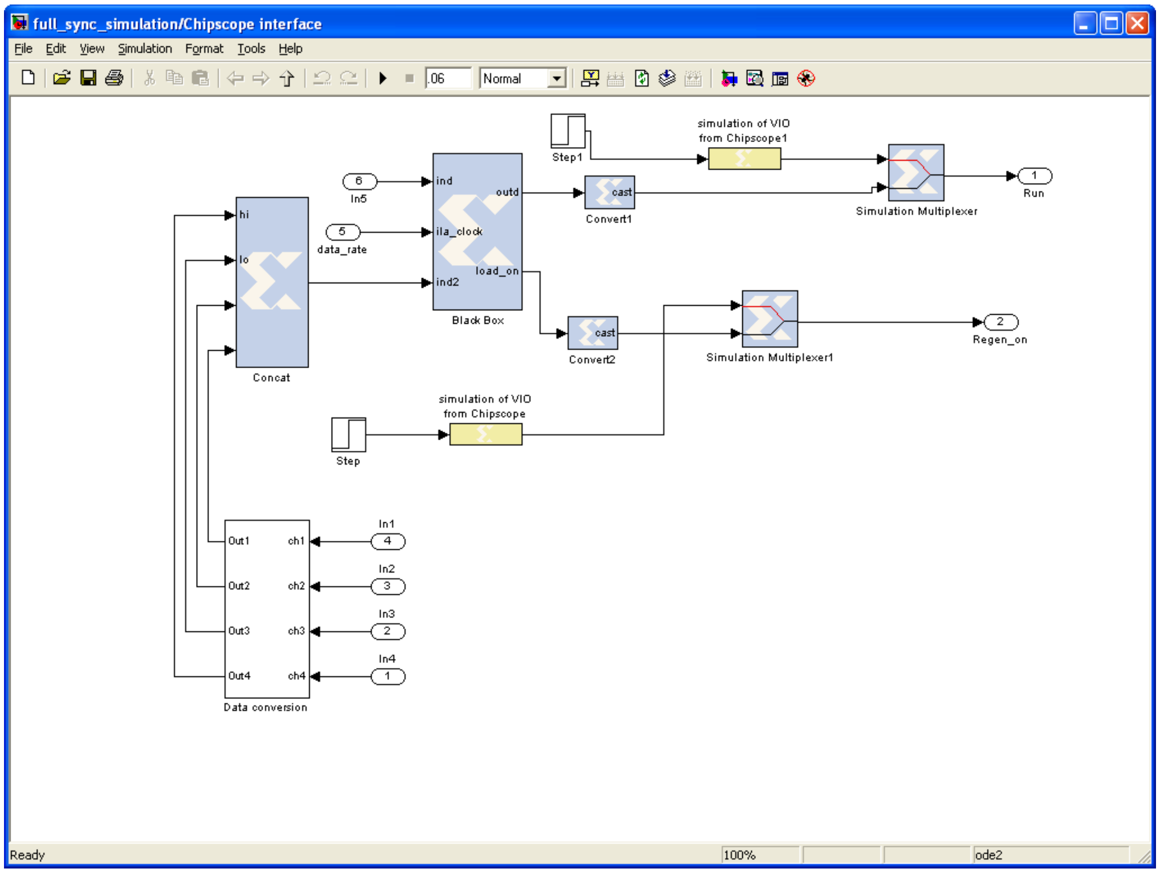

full_sync_simulation/.../Modulation/Subsystem1
File Edit View Simulation Format Tools Help
.06
Normal
1
In1
and
z-0
1
Out1
2
In2
not
Inverter
Ready
100%
ode2

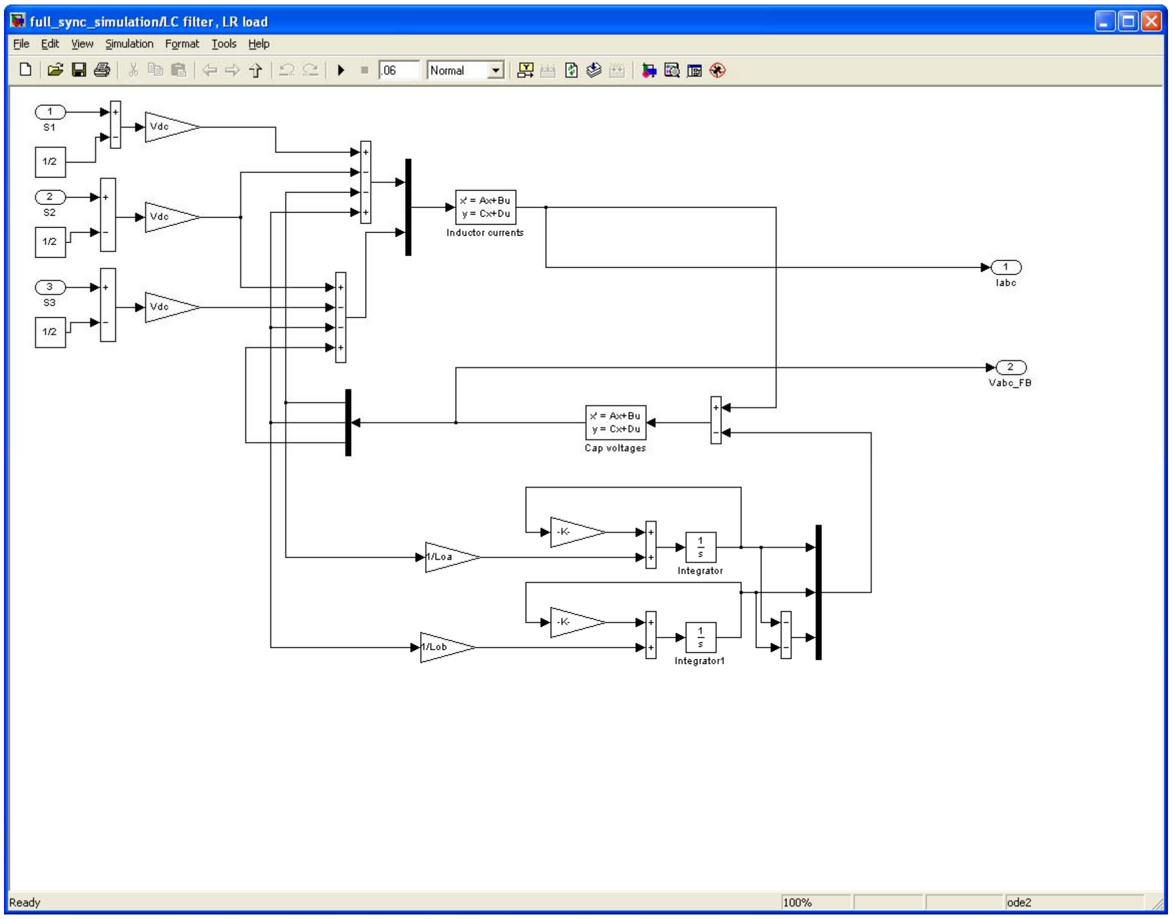

full_sync_simulation/Controller 2/Decoder
File Edit View Simulation Format Tools Help
.06
Normal
1
In1
Data_in
Data_out
Data_valid
Manchester_Decoder
Data
Data_valid
Data_in
Data_valid
Iq_Data
Id_Data
Vq_Data1
Vd_Data
Ring Decoder
z-1
Delay1
Delay
Delay2
Delay3
reinterpret
Reinterpret1
Reinterpret2
Reinterpret3
Reinterpret4
1
gain
2
gain1
3
gain2
4
gain3
Ready
100%
ode2


full_sync_simulation/.../Decoder/Manchester_Decoder
File Edit View Simulation Format Tools Help
.06
Normal
Data_in
Register3
Register4
Register6
Register
Register1
Register2
Register5
xlmax
MCode
x
y
z
Data_out
Data_valid
Ready
100%
ode2


full_sync_simulation/.../Decoder/Ring Decoder
File Edit View Simulation Format Tools Help
.06
Normal
1
Data_in
2
Data_valid
x
y
z
xlmax
MCode
Register8
Register9
Register1
Register3
Register10
Register6
Register7
Register2
cast
and
Constant
addr
data
we
1
Iq_Data
2
Id_Data
3
Vq_Data1
4
Vd_Data
Ready
100%
ode2


full_sync_simulation/Switching Unit
File Edit View Simulation Format Tools Help
.06
Normal
In1
In3
In5
In2
In4
In6
Shut off
COMM1
fault_in
SAin
SBin
SCin
FLTC
Logical1
Logical2
Logical3
Logical4
Logical5
Logical6
Logical7
Logical8
Logical9
Logical10
Logical11
Logical12
Logical
Inverter1
Inverter2
Inverter3
Inverter4
Inverter5
Inverter6
Inverter7
Mux1
Mux2
Mux
Mux4
Mux5
Mux3
Aplus U38
Bplus U40
Cplus U42
Aminus U39
Bminus U41
Cminus U43
Out1
Out2
Out3
Ready
100%
ode2


full_sync_simulation/Chipscope interface
File Edit View Simulation Format Tools Help
.06
Normal
Step1
simulation of VIO from Chipscope1
Simulation Multiplexer
1
Run
6
In5
ind
outd
cast
Convert1
5
data_rate
ila_clock
load_on
ind2
Black Box
hi
lo
Concat
cast
Convert2
Simulation Multiplexer1
2
Regen_on
simulation of VIO from Chipscope
Step
Out1
ch1
In1
4
Out2
ch2
In2
3
Out3
ch3
In3
2
Out4
ch4
In4
1
Data conversion
Ready
100%
ode2


full_sync_simulation/LC filter, LR load
File Edit View Simulation Format Tools Help
.06
Normal
S1
S2
S3
1/2
Vdc
Inductor currents
Iabc
Vabc_FB
Cap voltages
-K-
Integrator
Integrator1
1/Loa
1/Lob
Ready
100%
ode2

APPENDIX F: MATLAB CODE

A. INITIALIZATION FILE

```
Vdc=120;
dc=120;
Vref=50;

Kp_i=.01/3*(Vdc/sqrt(3));%current PI gain is amplified to account for
the SV modulation scaling
Ki_i=6*(Vdc/sqrt(3));    %Current control loop gain
Kp_v=0.15;
Ki_v=2.5;



Vdc_comp=30;
Vcesat=2.3;
delaycount=1;
oversample=1;  %1 4 work
fin=100;

tstop=40/60;
pulsect = 2400/oversample;
step_ct=1;
tstep = 40e-9*step_ct;
clkPeriod=tstep;
mod_index=.75;
F_mat = [0 0 0 1;1 1 2 0;2 2 3 0;3 3 0 0];
O_mat = F_mat;



s1=2*pi*1;
s2=2*pi*5000;
s3=2*pi*50000;
alpha=.0002*sqrt(3)/Vdc/2;



Lfa=350e-6;
Lfb=Lfa;
Lfc=Lfa;
Cf= 60e-6;
Cfa=Cf;Cfb=Cfa;Cfc=Cfb;
Loa=1e-4;
Lob=Loa;Loc=Loa;
Roa=20;
Rob=Roa;
Roc=Roa;

Amat_indI = -inv([Lfa -Lfb;Lfc Lfb+Lfc])*.005*[1 -1;1 2];
Bmat_indI = inv([Lfa -Lfb;Lfc Lfb+Lfc]);
```

```
Cmat_indI = [1 0 ;0 1 ;-1 -1 ];     %Ic = -Ia-Ib
Dmat_indI = zeros(3,2);

Amat_caps = zeros(3);
Bmat_caps = [1/Cfa 0 0; 0 1/Cfb 0; 0 0 1/Cfc];
Cmat_caps = eye(3);
Dmat_caps = zeros(3);

Amat_load = [-Roa/Loa 0 0; 0 -Rob/Lob 0; 0 0 -Roc/Loc];
Bmat_load = [1/Loa 0 0; 0 1/Lob 0 ; 0 0 1/Loc];
Cmat_load = eye(3);
Dmat_load = zeros(3);
```

B. INTERNAL CODE

1. thetaconv2.m

```
function [y] = thetaconv(x)
gain1 = xfix({xlSigned,14,10},2*3.14);
gain2 = xfix({xlSigned,14,10},1/gain1)
if x<0
y=xfix({xlUnsigned,10,0},(x+gain1)*gain2*1024);
else
y=xfix({xlUnsigned,10,0},x*gain2*1024);
end
```

2. overflow3.m

```
function [sector1, sector2, sector3, sector4, sector5, sector6, z] =
overflow3(x)
%gain = xfix({xlUnsigned,10,7},2.359296/3);%for 60 hz
gain = xfix({xlUnsigned,10,7},2.359296);%for 180 hz
%tempv=gain*x;
tempv=x;
if tempv<=171-1
    sector1=xfix({xlBoolean},1);
    sector2=xfix({xlBoolean},0);
    sector3=xfix({xlBoolean},0);
    sector4=xfix({xlBoolean},0);
    sector5=xfix({xlBoolean},0);
    sector6=xfix({xlBoolean},0);
    z=xfix({xlUnsigned,10,0},tempv);
elseif tempv<=2*171-1
    sector1=xfix({xlBoolean},0);
    sector2=xfix({xlBoolean},1);
    sector3=xfix({xlBoolean},0);
    sector4=xfix({xlBoolean},0);
    sector5=xfix({xlBoolean},0);
    sector6=xfix({xlBoolean},0);
    z=xfix({xlUnsigned,10,0},tempv-171);
elseif tempv<=3*171-1
```

```
    sector1=xfix({xlBoolean},0);
    sector2=xfix({xlBoolean},0);
    sector3=xfix({xlBoolean},1);
    sector4=xfix({xlBoolean},0);
    sector5=xfix({xlBoolean},0);
    sector6=xfix({xlBoolean},0);
    z=xfix({xlUnsigned,10,0},tempv-2*171);
elseif tempv<=4*171-1
    sector1=xfix({xlBoolean},0);
    sector2=xfix({xlBoolean},0);
    sector3=xfix({xlBoolean},0);
    sector4=xfix({xlBoolean},1);
    sector5=xfix({xlBoolean},0);
    sector6=xfix({xlBoolean},0);
    z=xfix({xlUnsigned,10,0},tempv-3*171);
elseif tempv<=5*171-1
    sector1=xfix({xlBoolean},0);
    sector2=xfix({xlBoolean},0);
    sector3=xfix({xlBoolean},0);
    sector4=xfix({xlBoolean},0);
    sector5=xfix({xlBoolean},1);
    sector6=xfix({xlBoolean},0);
    z=xfix({xlUnsigned,10,0},tempv-4*171);
else
    sector1=xfix({xlBoolean},0);
    sector2=xfix({xlBoolean},0);
    sector3=xfix({xlBoolean},0);
    sector4=xfix({xlBoolean},0);
    sector5=xfix({xlBoolean},0);
    sector6=xfix({xlBoolean},1);
    z=xfix({xlUnsigned,10,0},tempv-5*171);
end
```

3. ramp2mod.m

```
function z = ramp2(x)
gain=xfix({xlSigned,20,19},1/2400)
z=xfix({xlSigned,14,13},x*gain);
```

C. SERIALIZATION CODE

1. System Control Ring

```
function [Data_out, Index, Iq_en, Id_en, Vq_en, Vd_en, Busy, Send,
Counter] =
SystemControllerRingComm(Vd_Data_in,Id_Data_in,Vq_Data_in,Iq_Data_in,
Index_fb, Encoder_busy, Busy_fb, Iq_en_fb, Id_en_fb, Vq_en_fb,Vd_en_fb,
Counter_fb);

COUNTER_INITIAL_VALUE=100;  %%
```

```
%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Busy  %%
%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0) & (Encoder_busy==0) & (Counter_fb==0); %% Trigger
condition
    Counter= xfix({xlUnsigned, 7, 0},COUNTER_INITIAL_VALUE);
    Busy=xfix({xlUnsigned,1,0},1);
elseif (Busy_fb==1) & (Index_fb==10) & (Iq_en_fb==0) & (Id_en_fb==0) &
(Vq_en_fb==0) & (Vd_en_fb==0);
    Counter=xfix({xlUnsigned, 7, 0},Counter_fb-1);
    Busy=xfix({xlUnsigned,1,0},0);
elseif (Busy_fb==1)
    Counter= xfix({xlUnsigned, 7, 0},COUNTER_INITIAL_VALUE);
    Busy=xfix({xlUnsigned,1,0},1);
else
    Counter=xfix({xlUnsigned, 7, 0},Counter_fb-1);
    Busy=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Send  %%
%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0) & (Encoder_busy==0) & (Counter_fb==0); %% Trigger
condition
    Send=xfix({xlUnsigned,1,0},1);
elseif (Encoder_busy==1);  %% Disable if encoder is busy
    Send=xfix({xlUnsigned,1,0},0);
elseif (Busy_fb==1) & (Index_fb==11) & (Iq_en_fb==0) & (Id_en_fb==0) &
(Vq_en_fb==0) & (Vd_en_fb==0);
    Send=xfix({xlUnsigned,1,0},1);  %% End Case
elseif (Busy_fb==1);
    Send=xfix({xlUnsigned,1,0},1);
else
    Send=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Index  %%
%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0) & (Encoder_busy==0) & (Counter_fb==0); %% Trigger
condition
    Index=xfix({xlUnsigned,4,0},11);
elseif (Busy_fb==0)
    Index=xfix({xlUnsigned,4,0},0);
elseif (Encoder_busy==1)
    Index=xfix({xlUnsigned,4,0},Index_fb);
elseif (Index_fb>0) & (Encoder_busy==0)  %% Decrement
    Index=xfix({xlUnsigned,4,0},Index_fb-1);
elseif (Index_fb==0)
    Index=xfix({xlUnsigned,4,0},11);
else
    Index=xfix({xlUnsigned,4,0},0);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Iq_en  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0) & (Encoder_busy==0) & (Counter_fb==0)  %%trigger Case
    Iq_en=xfix({xlUnsigned,1,0},1);
elseif (Busy_fb==0)
    Iq_en=xfix({xlUnsigned,1,0},0);
elseif (Index_fb>0) & (Iq_en_fb==1)
    Iq_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Iq_en_fb==1) & (Encoder_busy==0)
    Iq_en=xfix({xlUnsigned,1,0},0);
elseif (Index_fb==0) & (Iq_en_fb==1) & (Encoder_busy==1)
    Iq_en=xfix({xlUnsigned,1,0},1);
else
    Iq_en=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Vq_en %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0);
    Vq_en=xfix({xlUnsigned,1,0},0);
elseif (Index_fb==0) & (Encoder_busy==0) & (Iq_en_fb==1)%% Start Vq_en
    Vq_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb>0) & (Vq_en_fb==1); %% the rest > 0
    Vq_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Encoder_busy==1) & (Vq_en_fb==1) %% last case
    Vq_en=xfix({xlUnsigned,1,0},1);
else
    Vq_en=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Id_en %%
%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0);
    Id_en=xfix({xlUnsigned,1,0},0);
elseif (Index_fb==0) & (Encoder_busy==0) & (Vq_en_fb==1)%% Start Id_en
    Id_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb>0) & (Id_en_fb==1); %% the rest > 0
    Id_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Encoder_busy==1) & (Id_en_fb==1) %% last case
    Id_en=xfix({xlUnsigned,1,0},1);
else
    Id_en=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Vd_en    %%
%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0);
    Vd_en=xfix({xlUnsigned,1,0},0);
elseif (Index_fb==0) & (Encoder_busy==0) & (Id_en_fb==1)%% Start Vd_en
    Vd_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb>0) & (Vd_en_fb==1); %% the rest > 0
```

```
    Vd_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Encoder_busy==1) & (Vd_en_fb==1) %% last case
    Vd_en=xfix({xlUnsigned,1,0},1);
else
    Vd_en=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Data out    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (Busy_fb==0)  %% Start bit
    Data_out=xfix({xlUnsigned,1,0},0);
elseif(Busy_fb==1)
    if (Iq_en_fb==1)
        if (Index_fb == 11)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,11,11));
        elseif (Index_fb == 10)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,10,10));
        elseif (Index_fb == 9)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,9,9));
        elseif (Index_fb == 8)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,8,8));
        elseif (Index_fb == 7)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,7,7));
        elseif (Index_fb == 6)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,6,6));
        elseif (Index_fb == 5)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,5,5));
        elseif (Index_fb == 4)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,4,4));
        elseif (Index_fb == 3)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,3,3));
        elseif (Index_fb == 2)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,2,2));
        elseif (Index_fb == 1)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,1,1));
        else
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Iq_Data_in,0,0));
        end

    elseif (Vq_en_fb==1)
        if (Index_fb == 11)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,11,11));
        elseif (Index_fb == 10)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,10,10));
        elseif (Index_fb == 9)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,9,9));
        elseif (Index_fb == 8)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,8,8));
        elseif (Index_fb == 7)
```

```
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,7,7));
        elseif (Index_fb == 6)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,6,6));
        elseif (Index_fb == 5)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,5,5));
        elseif (Index_fb == 4)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,4,4));
        elseif (Index_fb == 3)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,3,3));
        elseif (Index_fb == 2)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,2,2));
        elseif (Index_fb == 1)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,1,1));
        else
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vq_Data_in,0,0));
        end

    elseif (Id_en_fb==1)
        if  (Index_fb == 11)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,11,11));
        elseif (Index_fb == 10)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,10,10));
        elseif (Index_fb == 9)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,9,9));
        elseif (Index_fb == 8)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,8,8));
        elseif (Index_fb == 7)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,7,7));
        elseif (Index_fb == 6)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,6,6));
        elseif (Index_fb == 5)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,5,5));
        elseif (Index_fb == 4)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,4,4));
        elseif (Index_fb == 3)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,3,3));
        elseif (Index_fb == 2)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,2,2));
        elseif (Index_fb == 1)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,1,1));
        else
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Id_Data_in,0,0));
        end

    elseif (Vd_en_fb==1)
        if (Index_fb == 11)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,11,11));
        elseif (Index_fb == 10)
            Data_out =
xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,10,10));
        elseif (Index_fb == 9)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,9,9));
        elseif (Index_fb == 8)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,8,8));
        elseif (Index_fb == 7)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,7,7));
        elseif (Index_fb == 6)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,6,6));
        elseif (Index_fb == 5)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,5,5));
        elseif (Index_fb == 4)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,4,4));
        elseif (Index_fb == 3)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,3,3));
        elseif (Index_fb == 2)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,2,2));
        elseif (Index_fb == 1)
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,1,1));
        else
            Data_out = xfix({xlUnsigned,1,0},xl_slice(Vd_Data_in,0,0));
        end



    elseif (Index_fb==11)
        Data_out = xfix({xlUnsigned,1,0},1);  %% Stop Bit
    else
        Data_out=xfix({xlUnsigned,1,0},1);
    end
else
    Data_out=xfix({xlUnsigned,1,0},0);
end
```

2. Manchester Encoder

```
function [Counter,Data_out, Busy] = ManchesterEncoder(Counter_fb,
Data_in, Send, Data_fb, Busy_fb)

COUNTER_INITIAL_VALUE =24;
HALF_OUTPUT_WIDTH = 13;

%%%%%%%%%%%%%%%%%%%%%
%% Logic for Busy  %%
%%%%%%%%%%%%%%%%%%%%%
if (Send == 1) & (Busy_fb == 0);
    Busy = xfix({xlUnsigned, 1, 0},1);
elseif (Busy_fb == 1) & (Counter_fb > 2)  %count to 2 instead of zero
    Busy = xfix({xlUnsigned, 1, 0},1);
else
    Busy= xfix({xlUnsigned, 1, 0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%
%% Logic for Data_out %%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%
if (Send == 1) & (Busy_fb == 0) & (Data_in == 0) %%Start sending a 0->1
tran
    Data_out=xfix({xlUnsigned, 1, 0},0);
elseif (Send == 1) & (Busy_fb == 0) & (Data_in == 1) %%Start sending a
0->tran
    Data_out=xfix({xlUnsigned, 1, 0},1);
elseif (Busy_fb == 1) & (Counter_fb == HALF_OUTPUT_WIDTH) %%Toggle
output
    Data_out=xfix({xlUnsigned, 1, 0}, (Data_fb +1));
else
    Data_out=xfix({xlUnsigned, 1, 0}, Data_fb);
end

%%%%%%%%%%%%%%%%%%%%%%%%
%% Logic for Counter  %%
%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb == 0)
    Counter= xfix({xlUnsigned, 5, 0},COUNTER_INITIAL_VALUE);
elseif (Counter_fb==0)
    Counter= xfix({xlUnsigned, 5, 0},COUNTER_INITIAL_VALUE);
else
    Counter=xfix({xlUnsigned, 5, 0},Counter_fb-1);
end
```

3. Manchester Decoder

```
function [Busy, Data_out, Counter, Data_valid,Data_fil] =
ManchesterDecoder (Busy_fb, Data_Delayed, Data_in, Data_out_fb,
Counter_fb,In_delay1,In_delay2)

COUNTER_INITIAL_VALUE=16;  %%

%%%%%%%%%%%%%%%%%%%%%%%%%   Checks that three samples in a row
transition
%%%%%%%%%%%%%%%%%%%%%%%%%   before changing the input data
%%  Filter for Data_in  %%
%%%%%%%%%%%%%%%%%%%%%%%%%
if (Data_in == 0) & (In_delay1 == 0) &(In_delay2 == 0) &(Data_Delayed
== 0);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 0) & (In_delay1 == 0) &(In_delay2 == 0)
&(Data_Delayed == 1);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 0) & (In_delay1 == 0) &(In_delay2 == 1)
&(Data_Delayed == 0);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 0) & (In_delay1 == 0) &(In_delay2 == 1)
&(Data_Delayed == 1);
    Data_fil=xfix({xlUnsigned, 1, 0},1);
```

```
elseif (Data_in == 0) & (In_delay1 == 1) &(In_delay2 == 0)
&(Data_Delayed == 0);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 0) & (In_delay1 == 1) &(In_delay2 == 0)
&(Data_Delayed == 1);
    Data_fil=xfix({xlUnsigned, 1, 0},1);
elseif (Data_in == 0) & (In_delay1 == 1) &(In_delay2 == 1)
&(Data_Delayed == 0);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 0) & (In_delay1 == 1) &(In_delay2 == 1)
&(Data_Delayed == 1);
    Data_fil=xfix({xlUnsigned, 1, 0},1);
elseif (Data_in == 1) & (In_delay1 == 0) &(In_delay2 == 0)
&(Data_Delayed == 0);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 1) & (In_delay1 == 0) &(In_delay2 == 0)
&(Data_Delayed == 1);
    Data_fil=xfix({xlUnsigned, 1, 0},1);
elseif (Data_in == 1) & (In_delay1 == 0) &(In_delay2 == 1)
&(Data_Delayed == 0);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 1) & (In_delay1 == 0) &(In_delay2 == 1)
&(Data_Delayed == 1);
    Data_fil=xfix({xlUnsigned, 1, 0},1);
elseif (Data_in == 1) & (In_delay1 == 1) &(In_delay2 == 0)
&(Data_Delayed == 0);
    Data_fil=xfix({xlUnsigned, 1, 0},0);
elseif (Data_in == 1) & (In_delay1 == 1) &(In_delay2 == 0)
&(Data_Delayed == 1);
    Data_fil=xfix({xlUnsigned, 1, 0},1);
elseif (Data_in == 1) & (In_delay1 == 1) &(In_delay2 == 1)
&(Data_Delayed == 0);
    Data_fil=xfix({xlUnsigned, 1, 0},1);
else
    Data_fil=xfix({xlUnsigned, 1, 0},1);
end

%%%%%%%%%%%%%%%%%%%%
%%  Logic for Busy  %%
%%%%%%%%%%%%%%%%%%%%
if (Busy_fb == 0) & (Data_fil ~= Data_Delayed);
    Busy=xfix({xlUnsigned, 1, 0},1);
elseif (Busy_fb == 1) & (Counter_fb >0);
    Busy=xfix({xlUnsigned, 1, 0},1);
else
    Busy=xfix({xlUnsigned, 1, 0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Data_out and Data_valid  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb == 0) & ((Data_fil == 1) & (Data_Delayed == 0));  %%edge
detected
    Data_out=xfix({xlUnsigned, 1, 0},0);
    Data_valid=xfix({xlUnsigned, 1, 0},1);
```

```
elseif (Busy_fb == 0) & ((Data_fil == 0) & (Data_Delayed == 1));%%1->0
transition
    Data_out=xfix({xlUnsigned, 1, 0},1);
    Data_valid=xfix({xlUnsigned, 1, 0},1);
else
    Data_out=xfix({xlUnsigned, 1, 0},0);
    Data_valid=xfix({xlUnsigned, 1, 0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Counter  %%
%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb == 0) & (Data_fil ~= Data_Delayed);
        Counter=xfix({xlUnsigned, 6, 0},Counter_fb-1);
elseif (Busy_fb == 1) & (Counter_fb >0);
        Counter=xfix({xlUnsigned, 6, 0},Counter_fb-1);
else
        Counter = xfix({xlUnsigned, 6, 0},COUNTER_INITIAL_VALUE);
end
```

4. Ring Decoder

```
function [Data_out, Iq_en, Id_en, Vq, Vd_en, Busy, Index, Write,
Counter] = RingCommDecoder(Data_in, Data_valid, Iq_en_fb, Id_en_fb,
Vq_fb, Vd_en_fb, Busy_fb, Index_fb, Data_out_fb, Counter_fb)

START_BIT=0;
STOP_BIT=1;
DEADTIME_VALUE=40;  %%
WATCHDOG_VALUE=200;  %%

%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for CTR   %%
%%%%%%%%%%%%%%%%%%%%%%
if (Data_valid==1); %% data keeps resetting the counter
    Counter= xfix({xlUnsigned, 8, 0},0);
elseif (Counter_fb>WATCHDOG_VALUE)  %% watchdog fault condition
    Counter=xfix({xlUnsigned, 8, 0},Counter_fb+1);
else
    Counter=xfix({xlUnsigned, 8, 0},Counter_fb+1);
end

%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Busy  %%
%%%%%%%%%%%%%%%%%%%%%%
if (Data_in==START_BIT) & (Data_valid==1) & (Busy_fb==0) &
(Counter_fb>DEADTIME_VALUE)  %%Trigger condition
    Busy=xfix({xlUnsigned,1,0},1);
elseif (Busy_fb==0)
    Busy=xfix({xlUnsigned,1,0},0);
```

```
elseif (Data_in==STOP_BIT) & (Data_valid==1) & (Index_fb==11) &
(Iq_en_fb==0) & (Id_en_fb==0) & (Vq_fb==0) & (Vd_en_fb==0);  %%Stop
Case
    Busy=xfix({xlUnsigned,1,0},0);
elseif (Busy_fb==1)  %% Otherwise latch the  Busy_fb value
    Busy=xfix({xlUnsigned,1,0},1);
else
    Busy=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Index  %%
%%%%%%%%%%%%%%%%%%%%%%%
if (Data_in==START_BIT) & (Data_valid==1) & (Busy_fb==0) &
(Counter_fb>DEADTIME_VALUE)  %%Trigger condition
    Index=xfix({xlUnsigned,4,0},11);
elseif (Busy_fb==0)  %% Idle other than the trigger
    Index=xfix({xlUnsigned,4,0},0);
elseif (Data_valid==0) %% Data not valid
    Index=xfix({xlUnsigned,4,0},Index_fb);
elseif (Index_fb==11) & (Iq_en_fb==0) & (Id_en_fb==0) & (Vq_fb==0) &
(Vd_en_fb==0);  %%Stop Case
    Index=xfix({xlUnsigned,4,0},0);
elseif (Index_fb>0)  %% Decrement Condition
    Index=xfix({xlUnsigned,4,0},Index_fb-1);
elseif (Index_fb==0)
    Index=xfix({xlUnsigned,4,0},11);
else
    Index=xfix({xlUnsigned,4,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Iq_en  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Data_in==START_BIT) & (Data_valid==1) & (Busy_fb==0) &
(Counter_fb>DEADTIME_VALUE)  %%Trigger condition
    Iq_en=xfix({xlUnsigned,1,0},1);
elseif (Iq_en_fb==0)
    Iq_en=xfix({xlUnsigned,1,0},0);
elseif (Busy_fb==0)
    Iq_en=xfix({xlUnsigned,1,0},0);
elseif (Data_valid==0)
    Iq_en=xfix({xlUnsigned,1,0},Iq_en_fb);
elseif (Index_fb>0) & (Iq_en_fb==1);
    Iq_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Iq_en_fb==1)
    Iq_en=xfix({xlUnsigned,1,0},0);
else
    Iq_en=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Id_en  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0)
```

```
    Id_en=xfix({xlUnsigned,1,0},0);
elseif (Data_valid==0)
    Id_en=xfix({xlUnsigned,1,0},Id_en_fb);
elseif (Index_fb==0) & (Iq_en_fb==1)  %% First instance
    Id_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb>0) & (Id_en_fb==1);  %% Body
    Id_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Id_en_fb==1);  %%Stop case
    Id_en=xfix({xlUnsigned,1,0},0);
else
    Id_en=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Vq  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0)
    Vq=xfix({xlUnsigned,1,0},0);
elseif (Data_valid==0)
    Vq=xfix({xlUnsigned,1,0},Vq_fb);
elseif  (Index_fb==0) & (Id_en_fb==1) %% First instance
    Vq=xfix({xlUnsigned,1,0},1);
elseif (Index_fb>0) & (Vq_fb==1);  %% Body
    Vq=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Vq_fb==1);  %%Stop case
    Vq=xfix({xlUnsigned,1,0},0);
else
    Vq=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Vd_en  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0)
    Vd_en=xfix({xlUnsigned,1,0},0);
elseif (Data_valid==0)
    Vd_en=xfix({xlUnsigned,1,0},Vd_en_fb);
elseif (Index_fb==0) & (Vq_fb==1) %% First instance
    Vd_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb>0) & (Vd_en_fb==1);  %% Body
    Vd_en=xfix({xlUnsigned,1,0},1);
elseif (Index_fb==0) & (Vd_en_fb==1);  %%Stop case
    Vd_en=xfix({xlUnsigned,1,0},0);
else
    Vd_en=xfix({xlUnsigned,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Write
%%(Data_valid==1) &
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Data_valid==1) & (Index_fb==0) & (Busy_fb==1) & ((Iq_en_fb==1) |
(Id_en_fb==1) | (Vq_fb==1) | (Vd_en_fb==1));
    Write=xfix({xlBoolean},1);
else
```

```
    Write=xfix({xlBoolean,1,0},0);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Logic for Data_out    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (Busy_fb==0)
    Data_out=xfix({xlUnsigned,12,0},0);
elseif (Data_valid==0)
    Data_out=xfix({xlUnsigned,12,0},Data_out_fb);
elseif (Data_valid==1)
    if (Index_fb==11)
        Data_out=xfix({xlUnsigned,12,0},(2048 * Data_in));
    elseif (Index_fb==10)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (1024 *
Data_in));
    elseif (Index_fb==9)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (512 * Data_in));
    elseif (Index_fb==8)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (256 * Data_in));
    elseif (Index_fb==7)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (128 * Data_in));
    elseif (Index_fb==6)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (64 * Data_in));
    elseif (Index_fb==5)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (32 * Data_in));
    elseif (Index_fb==4)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (16 * Data_in));
    elseif (Index_fb==3)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (8 * Data_in));
    elseif (Index_fb==2)
        Data_out=xfix({xlUnsigned,12,0},Data_out_fb + (4 * Data_in));
    elseif (Index_fb==1)
        Data_out=xfix({xlUnsigned,12,0},(Data_out_fb + (2 * Data_in)));
    else
        Data_out=xfix({xlUnsigned,12,0},(Data_out_fb + Data_in));
    end
else
     Data_out=xfix({xlUnsigned,12,0},(Data_out_fb));
end
```

D. CHIPSCOPE CODE

```
function code_config(this_block)

  % Revision History:
  %
  %   11-May-2007  (09:32 hours):
  %     Original code was machine generated by Xilinx's System
Generator after parsing
  %     H:\Docs\work_files\faculty forms\lab
development\buck_converter\black_box_buck.vhd
```

```
  %
  %

  this_block.setTopLevelLanguage('VHDL');

  this_block.setEntityName('code');

  % System Generator has to assume that your entity  has a
combinational feed through;
  %   if it  doesn't, then comment out the following line:
  this_block.tagAsCombinational;

  this_block.addSimulinkInport('ind');
  this_block.addSimulinkInport('ila_clock');
  this_block.addSimulinkInport('ind2');

  this_block.addSimulinkOutport('outd');
  this_block.addSimulinkOutport('load_on');

  outd_port = this_block.port('outd');
  outd_port.setType('UFix_1_0');
  load_on_port = this_block.port('load_on');
  load_on_port.setType('UFix_1_0');

  % -----------------------------
  if (this_block.inputTypesKnown)
    % do input type checking, dynamic output type and generic setup in
this code block.

    if (this_block.port('ind').width ~= 1);
      this_block.setError('Input data type for port "ind" must have
width=1.');
    end

    this_block.port('ind').useHDLVector(false);

    if (this_block.port('ila_clock').width ~= 1);
      this_block.setError('Input data type for port "ila_clock" must
have width=1.');
    end

    this_block.port('ila_clock').useHDLVector(false);

    if (this_block.port('ind2').width ~= 48);
      this_block.setError('Input data type for port "ind2" must have
width=48.');
    end

  end  % if(inputTypesKnown)
  % -----------------------------

  % -----------------------------
   if (this_block.inputRatesKnown)
```

```
      setup_as_single_rate(this_block,'clk','ce')
    end  % if(inputRatesKnown)
  % ----------------------------



  % Add addtional source files as needed.
  %  |-------------
  %  | Add files in the order in which they should be compiled.
  %  | If two files "a.vhd" and "b.vhd" contain the entities
  %  | entity_a and entity_b, and entity_a contains a
  %  | component of type entity_b, the correct sequence of
  %  | addFile() calls would be:
  %  |    this_block.addFile('b.vhd');
  %  |    this_block.addFile('a.vhd');
  %  |-------------

  %    this_block.addFile('');
  %    this_block.addFile('');
  this_block.addFile('black_box_buck.vhd');

return;



% ------------------------------------------------------------

function setup_as_single_rate(block,clkname,cename)
  inputRates = block.inputRates;
  uniqueInputRates = unique(inputRates);
  if (length(uniqueInputRates)==1 & uniqueInputRates(1)==Inf)
    block.setError('The inputs to this block cannot all be constant.');
    return;
  end
  if (uniqueInputRates(end) == Inf)
     hasConstantInput = true;
     uniqueInputRates = uniqueInputRates(1:end-1);
  end
  if (length(uniqueInputRates) ~= 1)
    block.setError('The inputs to this block must run at a single
rate.');
    return;
  end
  theInputRate = uniqueInputRates(1);
  for i = 1:block.numSimulinkOutports
     block.outport(i).setRate(theInputRate);
  end
  block.addClkCEPair(clkname,cename,theInputRate);
  return;

% ------------------------------------------------------------
```

LIST OF REFERENCES

[1] MIL-STD-1399, "Interface standard for shipboard systems," Section 300A, October 1987.

[2] U. De Pra, D. Baert, and H. Kuyken, "Analysis of the degree of reliability of a redundant modular inverter structure," presented at the 20th international Telecommunications Energy Conference, San Francisco, October 1998.

[3] Debaprasad Kastha and Bimal K. Bose, "Investigation of fault modes of voltage-fed inverter system for induction motor drive," IEEE Trans. on Industry Applications, Vol. 30, No. 4, pp. 1028-1038, 1994.

[4] R. D. Klug and A. Mertens, "Reliability of megawatt drive concepts," presented at the IEEE international conference on Power System Technology, Vol. 2, pp. 636-641, 2003.

[5] Alexander L. Julian and Giovanna Oriti, "A comparison of redundant inverter topologies to improve voltage source inverter reliability," IEEE Trans. on Industry Applications, Vol. 43, No. 5, pp. 1371-1378, 2007.

[6] Benjamin S. Blanchard, Wolter J. Fabrycky, *Systems Engineering and Analysis*, pp. 372-380, Pearson Prentice Hall, Upper Saddle River, New Jersey, 2006.

[7] Alexander L. Julian, private conversation at Naval Postgraduate School, 15 November 2006.

[8] Paul C. Krause, Oleg Wasynczuk, and Scott D. Sudhoff, *Analysis of Electric Machinery and Drive Systems* pp. 113, IEEE Press, New York, 2002.

[9] Alexander L. Julian, Notes for EC4150 (Advanced Solid State Power Conversion), Naval Postgraduate School, Monterey, California, 2007 (unpublished).

[10] Alexander L. Julian, private conversation at Naval Postgraduate School, 5 June 2007.

[11] MEMEC^{TM} cooperation "Virtex-II^{TM} XC2V40/XC2V1000 Reference Board User's Guide" Version 2.0 July 2001.

[12] Semikron products market information http://www.semikron.com/internet/index.jsp?sekId=356 last visited 27 October 2007.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Jeffrey Knorr, Chairman, Department of Electrical and Computer Engineering
   Code EC/Ko
   Naval Postgraduate School
   Monterey, California

4. Dr. Alexander L. Julian, Department of Electrical and Computer Engineering
   Code EC/J1
   Naval Postgraduate School
   Monterey, California